

**פרויקט גמר הגנת הסייבר - FinalWall**

**נושא העבודה: חומת אש אפליקטיבית - WAF**

**שם תלמיד:** דניאל ספוז'ניקוב

**ת.ז תלמיד:** 215673369

**שם בית ספר ועיר :** קריית החינוך ע"ש עמוס דה-שליט, רחובות

**שם המנחה :** הילה גורן ברנע

**מועד הגשה :** 29/05/2024

תוכן עניינים

[1. מבוא 3](#_Toc168572316)

[2. תיאוריה 5](#_Toc168572317)

[3. תוצר סופי 13](#_Toc168572318)

[4. תהליך כתיבת הפרויקט 27](#_Toc168572319)

[5. מרכיבי פתרון 29](#_Toc168572320)

[6. תסריטי בדיקה 39](#_Toc168572321)

[7. רפלקציה 40](#_Toc168572322)

[8. הוראות התקנה ותפעול 42](#_Toc168572323)

[9. ביבליוגרפיה 43](#_Toc168572324)

[10. נספחים 44](#_Toc168572325)

1. מבוא
   1. נושא העבודה  
      הפרויקט שאני בחרתי ליישם במסגרת לימודי במגמת סייבר – תכנון, ופיתוח מערכות מידע הוא – Web application Firewall או כשמו המקוצר – WAF. את הפרויקט שלי ניתן למצוא בפלטפורמת Github אותו אני מכנה – FinalWall. מערכות WAF מהוות כחלק מרכזי במערך ההגנה על תוכנות Web אך הן לא היחידות, הן מצטרפות למגוון רחב של כלים אשר המטרה שהמשותפת שלהם היא להגן על האתר עליו הופקדו. תפקידה העיקרי של חומת אש אפליקטיבית היא לתחקר את התעבורה העוברת דרכה, לנטר אותה ולהציג אותה בדרך ידידותית. מערכת FinalWall מקנה ללקוח מגוון כלים לפיקוח התעבורה על תוכנת Web יחידה. כאשר ממוקמת בין לקוח לשרת, FinalWall יודעת לחקור את התעבורה אותה היא מקבלת ולנטר אותה בצורה יעילה ומהירה תוך שימוש במספר שיטות לזיהוי פקטות (חבילות מידע ברשת) חשובות. להלן השיטות ש-FinalWall משתמשת בהן:   
        
      **Core Rules Set** – קבוצת חוקים גנרים המשמשים לזיהוי מתקפות ברשת. חוקים אלו יכולים להיות מחרוזות או אפילו חוקי Regex. מדובר במרבד ההגנה הראשון של המערכת והבסיסי ביותר.   
      **Acess Control Lists** – רשימת כתובות אשר אינן יכולות לגשת לאתר. מורכבת מאינספור כתובות אשר נאספות הן מ-API שונים כמו ה-API של TOR והן מניתוח הודעות HTTP והוספת כתובות IP אשר מתקיפות את האתר באופן נפוץ.  
      **Progressive Ban System –** מערכת חסימות אושר בודקת לקוחות על פני היסטורית Requests. המערכת משתמשת בה כדי למתן תקשורת של צומת מסוים לאחר שעבר Threshold של הודעות בזמן מסוים או של מתקפות.  
      **Profiler –** יכולת ליצור פרופילים לכל לקוח וכך לשמור Forensics חשובים לצורך הצגה של הפרופיל בצורה הטובה ביותר.
   2. מטרות מרכזיות

המטרות היישומיות של הפרויקט הן:

* פיתוח מערכת יעודית אשר יודעת לא רק למצוא נוזקות ומתקפות ברשת אלא גם לדווח אותן בצורה יעילה וידידותית.
* ניתוח הודעות HTTP משני צדדי התקשורת (לקוח ושרת) וממשפחות שונות של מתקפות.
* ממש משתמש מתקדם מבוסס Web בעל 2 מערכות – Backend מבוסס FLASK ו-Frontend מבוסס Vue.js.
* פיתוח Package בפייתון אשר מאפשרת לכל מפתח ברחבי העולם ליצור WAF לוקלי או Remote.

המטרות האישיות של העבודה הן:

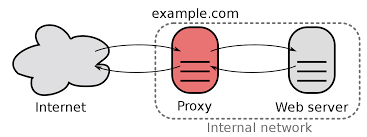
* למידה של מערכת ה-WAF לפרטי פרטים תוך שילוב חקירת מתקפות ברשת ויישום קוד אשר יודע לבלום אותן.
* ללמוד לפתח GUI מבוסס Web אשר יהיה מתקדם ויאפשר ללקוח למצות את יכולותיה של FinalWall בצורה מירבית.
* פיתוח פרויקט גדול ורציני הדורש עבודה המשכית ב- Github ובסטנדרטים הגבוהים של המגמה.
  1. רציונל  
     לאדם שאוהב פיתוח ואת כל מה שקשור לעולמות הרשת, חומת אש אפליקטיבית היא שילוב מדהים בין השניים. מצד אחד, יש בה את כל מה שאני אוהב ברשת – מחקר בנושא מתקפות ב-Web. יישום טכניקות הגנה מתקדמות, פיתוח פרופילים על לקוחות – Social Engineering. ומצד שני, פיתוח. תמיד אהבתי לפתח תוכנות ובדגש על רשתות ולכן בחרתי בפרויקט זה, מכיוון שהוא משלב את כל תחומי העניין שלי בצורה המיטבית ביותר. כמו כן, יש בפרויקט זה נושא מאוד חשוב והוא HTTP. מכיוון שמדובר בפרוטוקול כל כך חשוב ושיש לו חשיבות עצומה בעולם המודרני, רציתי ללמוד עליו הרבה יותר. לדעתי את כל הניואנסים שלו כשזה בא ל-Headers, Encryption ולחקור טכניקות WAF BYPASS שונות כמו הסתרת מידע עם שלבי Encapsulation שונים וכו. ניתן לומר שהמניע העיקרי בפרויקט הוא האהבה שלי לפיתוח מערכות סייבר מתקדמות הקשורות לווב.
  2. קישור לחומר הנלמד

הפרויקט מקושר לחומר הנלמד בכיתות י"א וי"ב במספר דרכים ונושאים.  
תחילה, במובן התקשורתי – המערכת בנויה על מספר שרתים בינהם שרת Frontend, Backend, שרת Tunneling אשר מאפשר תקשורת רציפה עם ה-Cluster ואת השרת Reverse Proxy עצמו של כל הCluster. כל שרת למעט שרתי הFrontend וה-Tunneling, הוא שרת אסינכרוני אשר בנוי על Low level API של סוקטים בספרייה Asyncio של פייתון. ה-Cluster עצמו מנהל תקשורת עם הלקוח והשרת Web ועושה זאת עם יישום High Level של Asyncio של Sockets. יישום זה נקרא Streams. ה- Cluster מנהל תקשורת עם כל צומת ברשת בצורה אסינכרונית המיושמת באובייקט – Stream.

בנוסף, היבט מערכות ההפעלה. FinalWall אינה מתעסקת בהיבטים שהם Low level אך היא מתפעלת שרתים ולכן בהינתן Connection מסוים כל שרת יפתח Coroutine אשר ינהל את התקשורת עם אותו Endpoint. ניתן להשוות Coroutine ל- Thread בכך שהוא משעה ומחדש את ביצוע המשימות שלו בסדר מסוים וכך מאפשר מקביליות בין המשימות עליו הוא אחרי. כמו כן, שרת הProxy של ה-WAF עצמו ושרת ה-Backend מנהלים Threads מקבילים ל-Main Thread.

1. תיאוריה
   1. תיאוריה  
      המטרה העיקרית של FinalWall היא לחסום מתקפות ולהתריע אותן למשתמש בצורה שהיא Real Time, מהירה ויעילה. המערכת עושה זאת על ידי חקירת הודעות HTTP אשר היא תופסת בעת גלישת לקוח לאתר אשר היא מגינה עליו. תחילה, אפרט על הידע הנדרש כדי להבין את מבנה המערכת ואחר כך אסביר את הידע הדרוש כדי להבין כיצד המערכת יודעת אילו הודעות יש לחסום.  
        
      **רשת מקומית (LAN)** – רשת מחשבים או רכיבי רשת בקנה מידה קטן אשר בנויה בסביבות ספציפיות כמו הבית, בית ספר, בניין חברה וכו.

**רשת ציבורית (WAN)** – רשת מרוחקת המחברת מחשבים ברשתות מקומיות אזוריות שונות או רשתות אזור מטרופולין לצורך תקשורת.   
  
**כתובת IP** – כתובת IP הינה אוסף של 4 בתים המפורדים עם נקודות כאשר כל בית מייצג מספר מסוים. בעזרת תווית מספרית זו ניתן לזהות רכיבי רשת שונים. ניתן לקטלג כתובת זו לכמה כתובת שונות אשר מביאות עימן טרמינולוגיה אחרת. **כתובת פרטית:** כתובת פרטית היא כתובת IP המייצגת רכיב רשת בתוך ה-LAN. אשר רק שאר רכיבי הרשת האחרים (בינהם הנתב) מכירים אותה. לרוב הן מתחילות ב-Prefix שמור. **כתובת ציבורית:** כידוע, כדי לגשת לאינטרנט על רכיב הרשת לעבור דרך הנתב. כתובת ציבורית היא כתובת המייצגת קבוצת כתובת IP פרטיות. היא מייצגת את כתובת המארח של של האינטרנט באיזור מסוים – Partner, Bezeq וכו.  
  
**שרת Proxy** – שרת אשר נמצא בתווך התקשורת של הלקוח והשרת הרצוי כך שהתקשורת עוברת דרכו. בשרתי פרוקסי רגילים הלקוח מודע לכתובת של השרת.

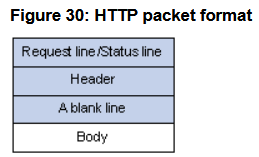
**שרת Reverse Proxy** – בעולם של היום, משלל סיבות ואינטרסים של בני אדם פותחו מספר שיטות ליישום ה-Proxy. ניתן לקטלג אותם לכמה סוגים אך מכיוון ש-WAF הינו Reverse Proxy אפרט עליו. סוג פרוקסי זה הוא שרת פרוקסי רגיל אך בניגוד לשרת פרוקסי רגיל, הוא מוסווה מפני הלקוח כך שמצד הלקוח מדובר בשרת היעד שלו.

**API** – קיצור ל- Application Programming Interface, API הם הדרך של מספר תוכנות לתקשר אחד עם השני. מדובר בכלי המשרת את שני קצוות התקשורת. פיתוח שרת API מאפשר לתוכנות חיצוניות לגשת למשאבים של שרת ה-Backend וכך לתקשר עימו ולהשתמש במידע שלו.

**חקירת הודעות HTTP:URI** – אוסף של תווים המזוהיים עם משאב אינטרנטי יחיד. (באופן חד חד ערכי).  
יש לו מבנה סינטקס גנרי והוא:  
   
URI = scheme ":" ["//" authority] path ["?" query] ["#" fragment]  
  
לכל חלק יש משקל נרחב בתהליך זיהוי הודעות HTTP מסוכנות ברשת. דוגמאות:   
  
1) **https://www.example.com/index.html  
2) http://localhost:5001/products?pid=acbdef1**   
  
  
**URL** – תת סוג של URI. נראה בדיוק כמו URI אך מטרתו היא לאפיין **היכן** יושב אותו משאב שה-URI מתאר, בניגוד ל-URI שמטרתו היא לאפיין כל משאב ברשת.

**HTTP** – קיצור ל-Hyper Text Transfer Protocol, HTTP מהווה כאבן יסוד ל-World Wide Web ומאפשר לצמתים שונים ברשת לשתף, לטעון ולהציג Webpages בבראוזר.  
פרוטוקול זה נמצא בשכבה 7 במודל ה-OSI. **תקשורת ב-HTTP** – תקשורת בפרוטוקול זה כוללת Request הנשלח מהלקוח ו-Response המוחזר משרת HTTP. מדובר בפרוטוקול חסר מצבים (Stateless) מה שאומר שהמידע של הודעת HTTP מסוימת לא נקשר לשום הודעה אשר באה לפניה. **שרת HTTP** – שרת אשר היעוד שלו הוא לשמור משאבי אינטרנט כמו דפי HTML, תמונות, וידיאו ועוד. ויודע להבין הודעות מסוג HTTP ואת השרשור שלהם ולהחזיר להם תשובה מתאימה.

איור 1 – מבנה הודעת HTTP

בפניכם השרשור המלא לפי ה-RFC של HTTP. ה-Request Line הינה שורת הפתיחה של ההודעה והיא מכילה את סוג ההודעה (Get, Post, Delete, Options וכולי). את ה-URI המבוקש. ואת הגרסה של הפרוטוקול. חלק ה-Header באיור מתייחס לשלל ה-Headers הקיימים ב-RFC של הפרוטוקול אשר הם מביאים מידע נוסף על כל מני היבטים של התקשורת בין הלקוח לשרת. Body הינו חלק בהודעה אשר נושא עימו מידע נוסף אשר הוזרק על ידי הלקוח או על ידי השרת. כל שורה מסתיימת בסימן – CRLF או ‘\r\n’. וחלק ה-Headers וה-Body מופרדים עם פעמיים CRLF. **מדובר במבנה של ה-RFC ואין לעבור עליו.**

ניתן "לעבור" על החוקים ולבצע מתקפת WEB על אתר בכמה דרכים תוך שימוש בפרוטוקול HTTP. FinalWall מגינה על האתר מפני מספר חולשות או מניעים חברתיים אשר מספיקים לקטלג לקוחות כמסוכנים בניהם: Cross Site Scripting, Sql Injection, כניסה לא מורשת, נסיון תקשורת מכלים לא חוקיים כגון טור או ממדינות שקשה מאוד לסמוך עליהן, לצורך העניין – איראן. להלן החומר התיאורתי.

**חולשה** – חור במערת מסוימת (יכול להיגרם מעיצוב תוכנה שגוי או יישום רע) אשר מאפשרים למתקיף לנצל את החולשה הזו במטרה לגרום הרס לתוכנה (במקרה שלנו – אתר), לקדם מניעים פוליטים וכדומי.  
  
**מתקפת WEB** – מתקפה המתרחשת בסביבת ה-WEB. המתקיף מנצל חולשה הקיימת באתר ומתקיף אותו. מתקפות יכולות לגרום למספר תקריות לא רצויות כמו הכנסת מידע רגיש לאתר, גניבת מידע רגיש ממסד נתונים, הקרסת שירות האתר, שינוי התנהגות האתר.

**סוגי התקפות ברשת**קיימות אינספור מתקפות ברשת מכל מני סוגים – FinalWall מגן מ-3 מהן מתוך ה-OWASP Top 10. והן הזרקת קוד SQL, XSS, וBroken Access Control.

* **SQL Injection**  
  מתקפה זו מתארת מצב שבוא המתקיף מגלה חולשה או משתמש בה באתר על מנת  
  להזריק קוד בשפת SQL כדי לממש מטרה זדונית. קורה כאשר הלקוח (המתקיף) מכניס מידע רגיש לאתר על ידי פורומים, לוגינים ועוד. מתקפת SQL Injection מוצלחת מתבטאת בכמה היבטים – נסיון מוצלח והזרקת קוד זדוני למסד הנתונים הגורמים להדלפת המידע במסד הנתונים (SELECT), שינוי המידע במסד הנתונים (INSERT\UPDATE\DELETE). ביצוע פעולות המורשות רק על ידי ה-Authority של האתר. או במקרים קיצוניים יותר להזריק Commands ל-Shell של מערכת הפעלה.  
  מתקפה זו יכולה לבוא לידי ביטוי בכמה מקרים: Login, העלאות קבצים, פורומים, פרמטרים ב-URL ועוד שלל אפשרויות אשר כוללות קליטת קלט מהמשתמש. ניקח לצורך הדוגמה את ה-Payload הנ"ל שהוא SQL Query לא חוקי: ‘OR ‘a’=’a’;--.  
  בהנחה שהמסד נתונים רגיש ל-SQLi, המתקיף יכול להזריק את המחרוזת הזו וכך לגרום למסד הנתונים להתעלם משאר הCredentials בLogin. (בגל צירוף התווים ;-- בסוף).  
  זוהי רק דוגמה אחת לאינספור קומבינציות של SQL שניתן לבנות כדי להתקיף אתר המתבסס מסד נתונים של SQL. ניתן לעשות מספר פעולות כמו הזרקת מידע רגיש לדוגמה:  
  ‘); DELETE \* FROM users;-- אשר תמחק את כל מסד הנתונים. כמו כן ניתן לעדכן עם UPDATE, או להכניס מידע חדש עם INSERT. אלו טכניקות שתוקפים צוברים לאט לאט ולכן על ה-WAF לדעת להתמודד עם מצבים שכאלו.  
    
   באיור זה אנו רואים דוגמה לפורום בו המשתמש מזריק SQL במטרה לדלג על שלב בAuthentication וכך לקבל גישה לאתר או ל-Admin.

איור 2 – התקפת SQL Injection על פני דף Login.

* **Cross Site Scripting – XSS**מתקפת XSS מאופיינת בהזרקת קוד זדוני לאתר כדי שיגיע ל-End User אחר ויגרום לו להריץ אותו או בצורה דומה, לגרום לאתר להריץ אותו בעצמו. למתקפות אילו אין מגבלה והן יכולות להשיג תוצאות כמו Redirection לאתר שהוא זדוני, להריץ קובץ .exe לא חוקי, גניבת Cookies מהשתמש, Phishing ועוד. ישנן מספר דרכים שבהן ניתן לגשת לביצוע XSS, נתייחס ל-2 מהן:  
    
  **Reflected XSS -** ידועה גם בשם מתקפה לא מתמידה, מתקפה זו קוראת כאשר Script לא חוקי מוזרק לאתר ומורץ בצד הלקוח בעת טעינת ה-Response. תוקפים לרוב יבצעו מתקפה זו כאשר האתר מקבל קלט מהמשתמש וטוען אותו בעת ההחזרה. לכן, תוקפים יכולים לבנות קישורים בהם הם הזריקו Payload לא חוקי, לינק ל-Malware או קובץ לא רצוי וכך לגרום להם לרוץ בצד הלקוח מכיוון שהם לא מעובדים כמחרוזת אלא כקוד. דוגמה לקישור המכיל בתוכו XSS.  
  http://socialize.com?query=latest&news<\script%20src=”http://evil.com/malicious.js”   
  לאחר הפצת הלינק, לקוחות תמימים יכולו ללחוץ עליו ולגרום למתקיף להצליח במשימתו.  
    
  **Stored XSS** – ידועה גם בשם כמתקפה מתמידה, תת מתקפה זו יכולה להשיג את כל המטרות של חברתה ביותר נפוצה (Reflected) בדרך שונה. מתקפה זו יוצאת לפועל כאשר האקר מזריק Script זדוני לאתר והאתר שומר אותו ומטעין אותו ב-Requests הבאים של יוזרים אחרים. לדוגמה – Blog. מתקיף יכול בקלות להזריק Payload של XSS לאחד מהתגובות ב-Blog וכך בכל פעם שלקוח אחר יטען את אותו הבלוג הקוד יורץ. מתקפה זו שונה מ-Reflected מכיוון שהאקר מזריק את הקוד ומחכה שלקוחות אחרים יגשו אליו בניגוד לשניה שמנסה להפיץ את הPayload ולגרום לאחרים להריץ אותו. אחד מהפקטורים הגדולים במתקפה זו היא היכן המידע נשמר (כמו כן, מעובד כקוד ולא כמחרוזת). לכן המתקיף יעדיף להזריק את ה-Payload במקום שהוא High Traffic, ינסה אולי להזריק בURI עם גישות אדמיניסטרטיביות, לא ב-Sessions אלא ב-Db מסוים וכולי. כך המתקיף מבטיח לעצמו שההזרקה נשארת לטווח ארוך ותגיע לקהל רחב יותר.
* **Broken Access Control**המתקפה מתרחשת כאשר התוקף מנסה לגשת למאשבים אשר גישתם חסויה מבעוד מועד.  
  לדוגמה גישה ל-/admin נחשבת ברוב המקרים פרטית ולא תורשה ללקוחות רגילים. לרוב שיטה זו משולבת עם BruteForce כדי לעשות Fuzz לכל מני משאבים וכך לקוות לקבל גישה למשאב ספציפי. מתקפה זו יכולה להדליף מידע רגיש או הרשאות כך שמאחורי כל WAF חייב לעמוד מנגנון access control חזק.

**WAF Bypass Techniques**לאורך השנים האקרים פיתחו שיטות שהמטרה שלהם היא לעבור את מנגנון הבדיקות של ה-WAF. המתקיף יכול לבצע מספר שיטות על מנת לעבור את מנגנון הבדיקות.

**Signature Evasion –** תוקפים ברוב המקרים ינסו להסוות את המידע שלהם על ידי שימוש במנגנונים אשר מקדדים את המידע אשר הם מזריקים לאתר. היות שרוב מערכות ה-WAF כיום בודקות את התאמת ה-User Input לחתימות קיימות לפעמים ניתן לעבור על השלב הזה על ידי קידוד הקלט. סיבה נוספת ל-Signature Evasion שהיא שמרבית ה-WAFים כיום מביאים איתם מנגנון זיהוי חתימות מבוסס Regex ולכן הודעות לפני Decode יעברו ואשר לא. לדוגמה, במקום להזריק <script> ניתן להזריק - PHNjcmlwdD4=. לאחר מכן, כאשר ה-WAF מתחיל לחקור את ההודעה הוא ישים לב להודעה הנראת מוזר ואם אין לו יכולות קידוד הוא יעביר את ההודעה לשרת HTTP ששם הוא כמובן יעשה Decode לPayload ויריץ את התוכן המוזרק. ישנן מספר מוגבל אך גדול מאוד של שיטות כאלו בינהן URL Encoding, Base64 ועוד הרבה תקנים של קידודים אלו.  
לדוגמה ה- XSS Payload הבא <img src=x onerror=confirm()//> רצוי להישלח באופן הבא: Cimg%20src%3Dx%20onerror%3Dconfirm()%2F%2F%3E.  
  
**Obfuscation – Obfuscation או עמעום בעברית היא שיטה שבה האקרים משנים את נראות הקוד שהם מזריקים או כותבים כדי שיהיה קשה לניתוח.** מדובר ב**שיטת BYPASS המכילה את Signature Evasion אך היא נרחבת יותר ומטרתה היא לגרום לניתוח קשה יותר של נסיון ה-BYPASS ושל הקוד בפרט. בהקשר של -WAF, האקר יכול להזריק קודים זדוניים או להעלות קבצים לא חוקיים אשר כוללים קוד שהוא עבר עמעום. הדבר נעשה באמצעות שיטות מוזרות ולא יחודיות כמו Encoding בשיטות שונות, הוספת פרמטרים לא חשובים ל-URL בצורה מהונדסת היטב, הזרקת Comments, וגם מחיקת Comments, שימוש בפונקציות השפה – JS וSQL למשל. ניתן להשתמש ב-CHAR של SQL או ב-Alert של JS בצורות מוזרות ולא ידידותיות לקורא.   
  
Anonymizations – אם ה-WAF תופס אותך לפחות לא הזהות האמיתית....בתקופה האחרונה נעשה שימוש רחב בכלים אשר מספקים אנונימיות למשתמש כמו TOR ו-VPN. לרוב מתקיפים לא יתקפו מהרשת הלוקלית שלהם בבית אלא יגלשו לאתר מתוך פרופיל אנונימי וכך ימזערו את הסיכוי של ה-WAF לדעת את ה-IP האמיתי שלהם וכך להימנע מלהיכנס לרשימות BLACKLIST כמו ACL. תהליך אנונימיזציה מתחיל כאלה התעבורה של הלקוח עוברת דרך רשימת שרתי Proxy ידועים של חברה מסוימת או של שירות ציבורי מסוים.  
כל פרוקסי יעביר את ההודעה הלאה בחוות הפרוקסים עד שהיא תגיע ליעד – ה-WAF.  
כמוסכמה בעולם ה-HTTP ופרוקסי בפרט, כל שרת פרוקסי מוסיף Header הנקרא – XFF.  
או בפירוט – X-Forwarded-For אשר מצביע על שרשרת שרתים אשר היו אחראים להעברת הודעת ה-HTTP. כל פרוקסי מוסיף את ה-IP שלו וכך ניתן לדעת את התווך שבו ההודעה עברה. אך מתוך שיקולים של הבטחת מידע כל מערכת WAF מתייחסת ללקוח כאל התא הכי ימני ברשימה מכיוון שהוא הכי אמין היות ולא ניתן לשנות את ערכו אף פעם, לעומת כל שאר הכתובות אשר עליהן ניתן לבצע Obfuscation.**

* 1. מוצרים קיימים

ישנם הרבה מאוד חברות גדולות אשר מציאות שירות WAF כל אחת ביישום שלה.  
על WAF טוב להיות נגיש, ידידותי למשתמש, בעל יכולות חסימה גבוהות ועליו לעמוד בסטנדרטים בינלאומיים וכדי של – OWASP Top 10, וכמובן להקנות יכולות של קונפיגורציה אישית למשתמש.

Cloudflare WAF – אחד ממוצרי הבטחת המידע של Cloudflare אשר מציע מגוון יכולות ופיצרים בעת ה-Deployment שלו **בענן**. משתמש במנוע מופעל למידת מכונה אשר יודע ללמוד ולהשתפר על בסיס טרנזציות עבר. מוצר Enterprise לכל עניין.

ModSecurity – מנוע WAF מבוסס קוד פתוח בגיטהאב אשר פותח לראשונה על ידיIvan Ristic במטרה להיות מנוע פילטור של הודעות HTTP בעיקר לשרתי Apache. לאחר תקופה הגיע לבסוף לידיה של OWASP. ModSecurity יודעת להביא פלטפורמה

אשר בונה את החוקים שלה בשפה משלה הנקראת SecRules. כמו כן, היא בנויה על Module משלה.

|  |  |  |  |
| --- | --- | --- | --- |
| קריטריונים\בפרויקט | FinalWall | CloudFlare WAF | ModSecurity |
| Core Rule Set – סט חוקים מובנה במערכת. | קיים במערכת.ז | קיים במערכת, נקראים Managed Rules ויש API אשר מראה אותם. | קיים במנוע, חוקים משורשרים בSyntax של שפת SecRules. |
| יכולת הוספת חוקים דינמיים. | לא קיים במערכת. | קיים במערכת על ידי Custom Rules עם ממשק דמוי "שפה" של Cloudflare. ניתן ליצור חוקים בכמה Phases של האנליזציה. | ניתן ליצור חוקים חדשים רק בשפת SecRules ללא ממשק. |
| Source Blocking | מנגנון קיים וחזק במערכת. | מצהירים שיש אך אין בפועל. | אין פירוט. |
| יכולת זיהוי וחסימה של מתקפות ב-OWASP Top 10 | יודע לחסום 3 קטגוריות של תקיפה. Injection ו-Broken Access Control כאשר 2 מהן הם תת קטגוריות ב-Injection. | יודע לחסום את כל OWASP Top 10. | יודע לחסום את כל OWASP Top 10. |
| חסימת לקוחות על פי ACL של אנונימיזציה ושל מיקום גיאוגרפי. | אופציה קיימת. מבוסס Alpha-2 | אופציה קיימת. מבוסס Alpha-2. | אופציה קיימת. מבוסס Alpha-2. |
| נדרש תשלום | לא נדרש תשלום. | מכיוון שמדובר במוצר תעשייתי החברה דורשת תשלום Pro מינימלי של 20$ לחודש. | לא נדרש תשלום. |
| ממשק משתמש | יש ממשק משתמש. | יש ממשק משתמש. | אין ממשק משתמש. |
| רמת הממשק: תיאור תרחישים בצורה גרפית ידידותית, לוגים, Events, הצגת נתונים נוחה. | הממשק כולל לגרפיקה ידידותית – גרפי עמודות אשר עושות Map למספר המתקפות של כל IP, תרשים Doughnut של סך המתקפות ותרשים CPU כולל של המערכת. המערכת כוללת דף Events ומציג את הנתונים בעיצוב נוח לעין. כל הקוד הוא קוד פתוח ב-Github כולל -Readme למפתחים. | יש ממשק משתמש איכותי אשר כולל את כל התבחינים כולל דף Docs על ה-API של המערכת אשר עוזר למשתמש להשתמש ב-UI. | אין ממשק נוח למשתמש אך יש  רפוזיטורי קוד פתוח ב-Github אשר מאפשר לפתחים.  להבין כיצן להשתמש במנוע. |

1. תוצר סופי
   1. תיאור הפרויקט

מערכת FinalWall בנויה מכמה שכבות הגנה וארכיטקטורה. בכלליות המערכת תלויה בכמה גורמים שהם מנוע ה-WAF שבתוכו יש חקירת הודעות HTTP על גבי מספר חולשות: בדיקת SQLi, בדיקת XSS, בדיקת Access Control שאותה המערכת מכנה – Unauthorized Access בממשק המשתמש ובדיקת XFF מתוך ה-ACL של האנונימיזציה. המערכת עושה Caching לכל Events וללוגים המתאימים להם ומפתחת ומעדכנת את הפרופילים של הלקוחות ב-Real Time. כמו כן המערכת בנויה על תקשורת Client Server על מנת להעביר Events בין הWAF לAPI.

FinalWall כולה בנויה מכמה רכיבים עיקריים:

* מנהרה – Tunnel – תווך תקשורת בין ה-WAF ל-API ובו הוא מעביר את כל הEvents שנוצרים במערכת.
* שרת WAF – מנוע WAF אשר מתמקם בין הלקוח לשרת עליו הוא הופקד. מיושם בו שרת אסינכרוני פשוט ובעל 3 Threads. מנהל Thread ראשי ובוא הוא ממקם את השרת ומתחיל להקשיב לבקשות כאשר ברקע הוא מריץ 2 Threads אחרים: CPU Thread ו-ACL Thread. השרת מקשיב לבקשות וחוקר אותם, ולבסוך מדווח אותם לשרת ה-Frontend דרך API.
* שרת FinalWallAPI – שרת API המבוסס Flask. מנהל 2 משימות במקביל: משימת הרצת השרת העיקרי של Flask והעברת TunnelEvents מצידו של הWAF.
* שרת Frontend מבוסס Vue.js אשר מציג את המידע שהוא מקבל מ-API.

כאשר המערכת עולה לאוויר ב-3 שלבים: העלת שרת ה-WAF, לאחר מכן העלת ה-API ובשלב הסופי העלת ממשק המשתמש. המערכת יודעת להתגבר על מקרים מיוחדים כאשר שרת מסוים לא פועל. כאשר מפעילים את שרת ה-WAF הוא יודע לזהות מתי ה-API עולה לאוויר ושומר על כל ה-EVENTS שיש לו להודיע בתור. ולהפיך, גם ה-API יודע מתי ה-WAF ועולה ולכן לא יקרוס בעת ההרצה שלו.  
  
בעת כניסת המשתמש ל-UI יוצג לו ה-Dashboard ושם מידע כללי על המערכת.  
הוא יוכל לעיין במידע ואפילו להתשממשק עם הגרפים למשל, לבחור אילו מתקפות להציג בגרף ה-Distribution. מרגע העלת המערכת (יותר מדויק, מרגע העלת המנוע וה-API) הממשק פועל בצורה דינמית לחלוטין ו-RealTime כך שאין צורך בלחיצת ה-Refresh בכל Events. המשתמש יוכל לעיין בדפים נוספים ולראות את המידע מתעדכן בצורה חלקה. כאשר ה-API נופל הממשק יעצור מלקבל Events וימשיך להתעדכן כל עוד ה-API באוויר.

* 1. אלגוריתמים עיקריים
* **העלת Tunnel -** לשרת ה-WAF יש מנהרה שהיא תווך התקשורת שלו עם ה-API.  
  כאשר היא עולה לאוויר היא מנסה להתחבר לשרת המנהרה הראשי של הAPI שהוא פועל ב-Thread נפרד. השרת מבוסס שרת אסינכרוני של Websockets מעל תקשורת TCP ולכן גם ה-Tunnel של ה-WAF הינו מבוסס TCP WebSocket. בעת נסיון ה-Connect וכניסיון להתגבר על מקרים בהם ה-API אינו באוויר המנהרה תתחבר לשרת עד ל-Timeout מסוים וכאשר הוא נגמר הThread יפסיק. בזמן התרחיש של התחברות ה-WAF פועל ולכן הוא יכול לקבל בקשות, לחקור אותם ולטפל בהן ולכן עם הWAF "צריך" להודיע ל-API על Event במערכת אך הSocket עוד לא מחובר הוא ישמור את כל ה-Events בתור אסינכרוני ובעת ההתחברות הוא ישלח אותם בסדר הנדרש.
* **Health Monitoring -** כאשר ה-WAF עולה לאוויר הוא ישלח מדי שניה את מצב ה-CPU שלו ויעדכן את השרת.השרת שומר על CPU רק בדקה האחרונה ולכן כאשר יגיע ל-60 פעימות הוא יעשה popleft (יוצא את האיבר הראשון אשר מסמל את השניה הרחוקה ביותר – s1( וימשיך משם. כדי להימנע מRace Conditions הוא שולח את הEvent של הHealth תחת Lock.
* **פירוק הודעת HTTP פשוטה והפיכתה לטרנזציה –** בעת קבלת הודעות HTTP המיועדות לאתר המקורי ולשרתו המנוע יפרק את ההודעה לחלקים ויהפוך אותה לטרנזציה – אובייקט הבנוי ממספר שדות המקילים על תהליך הזיהוי של ההודעה כמסוכנת או לא. להלן השדות:  
  owner – בעל ההודעה מסוג HostAddress  
  real\_host – הלקוח האמיתי של ההודעה אשר מיובא מאלגוריתם ה-XFF של המערכת, מסוג HostAddress.  
  raw - מבנה ההודעה בשלמותה ב-bytes.  
  creation\_date – זמן מדויק בה הטרנזציה נקלטה במערכת מסוג מחרוזת או בתים.  
  מתודה – סוג המתודה של ההודעה של הלקוח. מסוג Method (מחלקה שעוטפת מחרוזת).   
  url – טיפוס שמחזיק פירוק של כל החלקים של הURL מסוג ParseResultBytes.  
  version – גרסת הפרוטוקול. מסוג מחרוזת או בתים.  
  query\_params – מילון אשר מחזיק בכל הפרמטרים של הURL שהלקוח הביא עימו.  
  ממפה query name לכל ערך. מסוג מילון.  
  headers – כל הHeaders של ההודעה. ממפה Header לערך שלו. מסוג מילון.  
  body – שדה שמחזיק בצורה שלמה את ה-body של ההודעה. מסוג bytes או מחרוזת.  
  size – משתנה מסוג interger שמחזיק את גודל ההודעה בבתים.

הפרויקט מיישם את הפירוק על פי ה-RFC של הפרוטוקול (HTTP) באופן עצמאי. ניתן לתאר את התהליך הזה לכמה תת תהליכים.

* + **עיבוד ה-Request Line:** כמפורט בRFC מספר 2616 של HTTP הRequest Line בנוי מ3 חלקים המופרדים בספרטור מסוים. כשההודעה מתקבלת הפעולה פונה לשורה הראשונה בהודעה ועושה split עם Seperator בנוי במערכת. פורמט הRequest Line: [ Method SP Request-URI SP HTTP Version CRLF ].   
    ה-Request URI עובר parsing של ספריית urllib שיודע לפרק את המחרוזת לחלקים המתאימים לה (scheme, params, path, netloc, query וFragment).  
    Method עובר לשדה המתודה והגרסה עוברת לשדה הגרסה.
  + **עיבוד גוף ההודעה והHeaders-:** מחשב את גודל ההודעה ולאחר מכן עושה Split לRaw עם ספרטור מובנה במערכת של Double CRLF - \r\n\r\n. לאחר כבר כאן מופעלת פעולה חשובה מאוד בתהליך ההמנעות מטכניקות Bypass שונות עליה אפרט באלגוריתם הבא אשר המטרה שלה היא לעשות Decode לכל Header ולכל Value שלו שנמצא.
  + **עיבוד משתני ה-Query params –** המערכת בודקת את המתודה של ההודעה ולפיה היא יודעת היכן למצוא את הפרמטרים. משתמשת בפעולה parse\_qs של Urllib ושוברת את המחרוזת הבסיסית של הURL לחלקים. לאחר מכן ממפה כל query לערכים שלו ועוברת תהליך Decoding שוב פעם כדי להימנע מטכניקות Bypass.
* **Decoding –** אלגוריתם פיענוח מחרוזת שיודע לפענח מחרוזות אשר מוצפנות או מעומעמות ב-Encoding Scheme מסוים. תומך ב- 5 Schemes נפוצים בעולם ה-Bypassing כמו Base64, URL Encoding ו-HTML Encoding ועוד.
* **Checking –** תהליך ה-Signature Comparison ושימוש ב-Core Rule Set המובנה במערכת. אפשר לפצל אותו ל2 קטגוריות שהן בדיקות לעומת ACL או בדיקות לעומת ה-CRS שהוא ה-Core Rule Set.
  + **בדיקות לעומת CRS –** בדיקות אלו כוללות בדיקות ל-Broken Access Control, Sql Injection ו-XSS.
    - **בדיקות ל-SQL Injection:** הבדיקה נעשת תחת Lock של מערכת ההפעלה כדי להימנע ב-Race Conditions כאשר כל Coroutine של השרת ניגש לCRS. תחילה יש בדיקה לתווים שיוצאים מההקשר של שרת הSQL. Context Escapers. האלגוריתם עובר על הפרמטרים של הURL ועל כל value של כל Header. ל-FinalWall לא משנה מתי הוא מופיע אך אם יופיע בתחילת האיזור הנבדק הוא יפסל ויחשב כנסיון פגיע במאשבי השרת. לאחר מכן, המערכת עוברת לבדיקת הHeader Values והפרמטרים כניגוד לCRS: יש 2 סוגי בדיקה – חתימות גנריות וחתימות גנריות משולבות. הבדיקה הגנרית בודקת על פי אופרטור in ותומכת בCase Sensitivity. כלומר המערכת יודעת לזהות נסיון עמעום – Obfusaction מסוג זה. בדיקה שניה – **Pair Checking**. בCRS יש חתימות גנריות שכאשר הן מופיעות לבד הן חוקיות בהחלט כמו Select (אנשים יכולים להחליט שהסיסמה שלהם הם select מכיוון שאין השפעה ישירה על הDB רק מלעשות Query ישיר מסוג זה) איך כאשר היא משולבת עם חתימות אחרות היא יכולה ליצור חור באתר ולגרום להשלכות חמורות. לרוב האקרים יעדיפו להשתמש קודם במילים מבודדות ולאחר מכן לשלב אותן עם עוד חתימות כדי להבין את רמת המורכבות של הWAF שעימו עם מתמודדים. בדיקת Pairs כוללת בדיקה של האופרטור in תוך תמיכה בCase Sensitivity כאשר היא מדווא שהחתימה הנוכחית אינה מופיעה עם אחת מחתימות איתה היא יכולה להשתלב ולסכן את האתר. במידה וההודעה מכילה payload שמכיל חתימה מה-CRS יוחזר Classifier של SQL Injection.
    - **בדיקות ל-XSS:** מתבצעת בדיקה עם האופרטור in כנגד כל החתימות ב-CRS גם בפרמטרים וגם ב- Headers. במידה וההודעה מכילה payload שמכיל חתימה מה-CRS יוחזר Classifier של XSS.
    - **בדיקה ל-Broken Access Control:**  FinalWall מתייג את הקטגוריה הזו כ-Unauthorized Access ומזהה נסיונות לגשת למאשבים לא מורשים עם שימוש באופרטור in. הוא בודק את הPath בURL של כל טרנזציה בשביל למצוא חתימות המייצגות משאבים אסורים בCRS.
  + **בדיקות אל מול ACL:** בדיקות אלו מתבצעות אל מול רשימת גישה של כתובות IP.  
    ישנן 3 ACL מרכזיים במערכת: **Generic ACL** שאוסף כתובות המתאפיינות בכך שלא ניתן לסמוך עליהן. דוגמה לכתובת כזו היא 2.58.56.43 שמזוהה כחלק מרשת טור ועל כך נכנסת ל-ACL בצורה דינמית. ה-ACL מחזיק בכתובות שהן מזוהות עם TOR או כל Vendor אחר של אנונימיזציה. ה-ACL מעודכן ב-Thread משלו כל חצי שעה על ידי גישה ל-API שונים אשר אפרט עליהם בהמשך. **ה-ACL השני** שמור במסד נתונים Redis (עליו אפרט בהמשך) ומאתר כתובות שמזוהות עם מספר התקפות גבוה מהרגיל. המערכת בולמת את אותה כתובת כמה וכמה פעמים עד שתגיע לסך התקפות (Threshold מסוים) כאשר הערך הדיפולטיבי הוא 3. לאחר מכן היא תכניס את אותה כתובת למסד הנתונים ותחסום אותה עוד פעם אם היא מופיע ב-ACL של החסימות שהן Persistent. (מתמידות). **ה-ACL השלישי** אוגר רשימת מדינות שלא ניתן לסמוך עליהן. כמו איראן, לבנון, סוריה, דרום קוריאה וכו. הוא משיג את אותן מדינות (קוד Alpha-2) מתוך מאגר בינלאומי של Maxmind. (אפרט עליו בהמשך). הבדיקה פשוטה: סיווג כתובת ה-IP למדינה אליה היא שייכת ובדיקת המדינה אל מול ה-ACL של המדינות. בדיקה זו תחזיר Classifier מסוג Banned Geolocation.
    - **בדיקת X-Forwarded-For:** כדי לזהות האם הלקוח משתמש בתוכנת אנונימיות המערכת מזהה את ה-Header של XFF (קיצור לX-Forwarded-For) ובודקת אל מול ה-ACL הגנרי את כל הכתובות הנמצאות בערכי ה-Header. במידה וכתובת נמצאת הלקוח נחסם תחת ה-Classifier של Anonymization. הבדיקת נעשה בצורה אסינכרונית כאשר כל כתובת IP בHeader עוברת עיבוד אשר בודקת את אותה כתובת אל מול ה-ACL אלה גם אל מול בדיקת כתובת חוקית. לא בכדי שהאקר ינסה לבדוק את ה-WAF העומד מולו בכך שישים ערכים ב-XFF שהם לא חוקיים. כמו מחרוזות סתמיות ואפילו Payloads של נוזקות.
    - **בדיקת כתובת IP:** בדיקה זו מתבצעת על ידי השוואת הכתובת לכתובת ב-ACL ובדיקת השרשור שלה ולבסוף השוואת המיקום הגיאוגרפי של אותה כתובת עם ה-ACL הגיאוגרפי. אם היא לא משורשרת היטב או נמצאת ב-ACL הכתובת נחשבת לכתובת מסוכנת. בדיקה זו נעשה על ידי בדיקת ה  
      check\_netloc והיא ממרימה את המחרוזת לכתובת IP, אם יש שגיאה תורם ה-Exception של ValueError וכך ניתן לדעת שמדובר במחרוזת לא חוקית ב-Header ה-XFF. לאחר בדיקת הכתובת אל מול ה-ACL הגנרי ובדיקת השרשור המערכת משיגה את המיקום הגיאוגרפי של הכתובת מתוך MaxMind API ובודקת את המיקום הגיאוגרפי אל מול הACL השלישי.
* **העברת מידע על Events במערכת:** אירועים במערכת מוגדרים תחת מחלקת ה-TunnelEvents שהיא אוגרת את כל האירועים האפשריים. כל אירוע מדווח ל-API בעזרת Decorator אשר פועל בסיום ריצת התחקיר או פעולה. ה-Event נשלח Tuple אשר אורז את סוג הEvent בתא הראשון ואת המידע הראוי בתא השני. לאחר שליחת האירוע מצד ה-WAF לשרת ה-Websockets של ה-API הראשי הוא מעביר באמצעות socketIO של Flask את האירוע ואת הראוי לו לשרת ה-Frontend. שרת זה מציב Listeners לכל Event ראוי כדי שידע לאיזה פונקציה לעשות Dispatch ב-Store במטרה לעדכן באופן אוטומטי את הממשק. Store הוא מאגר נתונים שבא עם vuex-js והמטרה שלו היא לאגור את המצב הנוכחי של הממשק. הוא כולל בתוכו מידע שמוצג ב-Dashboard, בלוח האירועים וב-Cluster Panel. המושג Dispatch מתאר שרשרת אירועים הכוללת מספר שלבים.  
  השלב הראשון הוא קריאה לפעולה Dispatch שתפעיל מנגון מסוג Action בStore. מנגון אשר עושה commit לMutation. השלב השני הוא שלב ה-Commit שהוא שלב שמשנה את המצב של ה-Store. מקבל את המידע החדש מהמערכת ומעדכן אותו.
* **בניית Security Page אישיותי לכל לקוח:** במערכת מוגדר קובץ קונפיגורציה מסוג TOML, אשר מגדיר את קבועי המערכת. כאשר המערכת מזהה לקוח מסוכן היא תפנה למחלקה אשר בונה דף HTML של חסימה התלוי בקבועים של המערכת. היא תודיע למתקיף שהוא חסום והמידע היחיד שיוצג הוא ה-Activity Code שלו אשר לא מסתמך על הבקשה של הלקוח אלא נבנה באופן רנדומלי.
  1. דרישות ומגבלות מערכת

השאיפה שלי היא שהמערכת תעמוד בסטנדרטים של המתחרים בשוק וזה אומר לדעת ליישם

רוב המנגנונים העיקריים של WAF בצורה מיטבית ועמידה. אי לכן יש כמה דרישות ואילוצי פתרון.

**בהיבט הדרישות:**

* המערכת נדרשת לתמוך במספר מתקפות ושיטות הגנה כגון Injection Attacks הכוללים XSS ו-SQL Injection, Broken Access Control וזיהוי כלי אנונימיזציה. POC על כל חולשה היא הבסיס למערכת טובה ולכן על המערכת להוכיח כי היא יכולה לזהות כל מתקפה ולסווג אותה כהלכה מבלי לשגות בתהליך הזיהוי ולספק מידע מדויק.
* המערכת נדרשת לתמוך ב-Source Blocking איכותי מתמיד שמתפשט על גבי 3 סוגים שונים של ACLs.
* על המערכת לזהות לדווח על אירועים באופן אירעי – Real Time בצורה המהירה ביותר.
* המערכת נדרשת לספק ממשק משתמש נוח וקל לשימוש כדי להקל על חקירת התעבורה.
* על המערכת לזהות נסיונות Bypass שונים הכוללים עמעום או Signature Evasion.

בהיבט המגבלות:

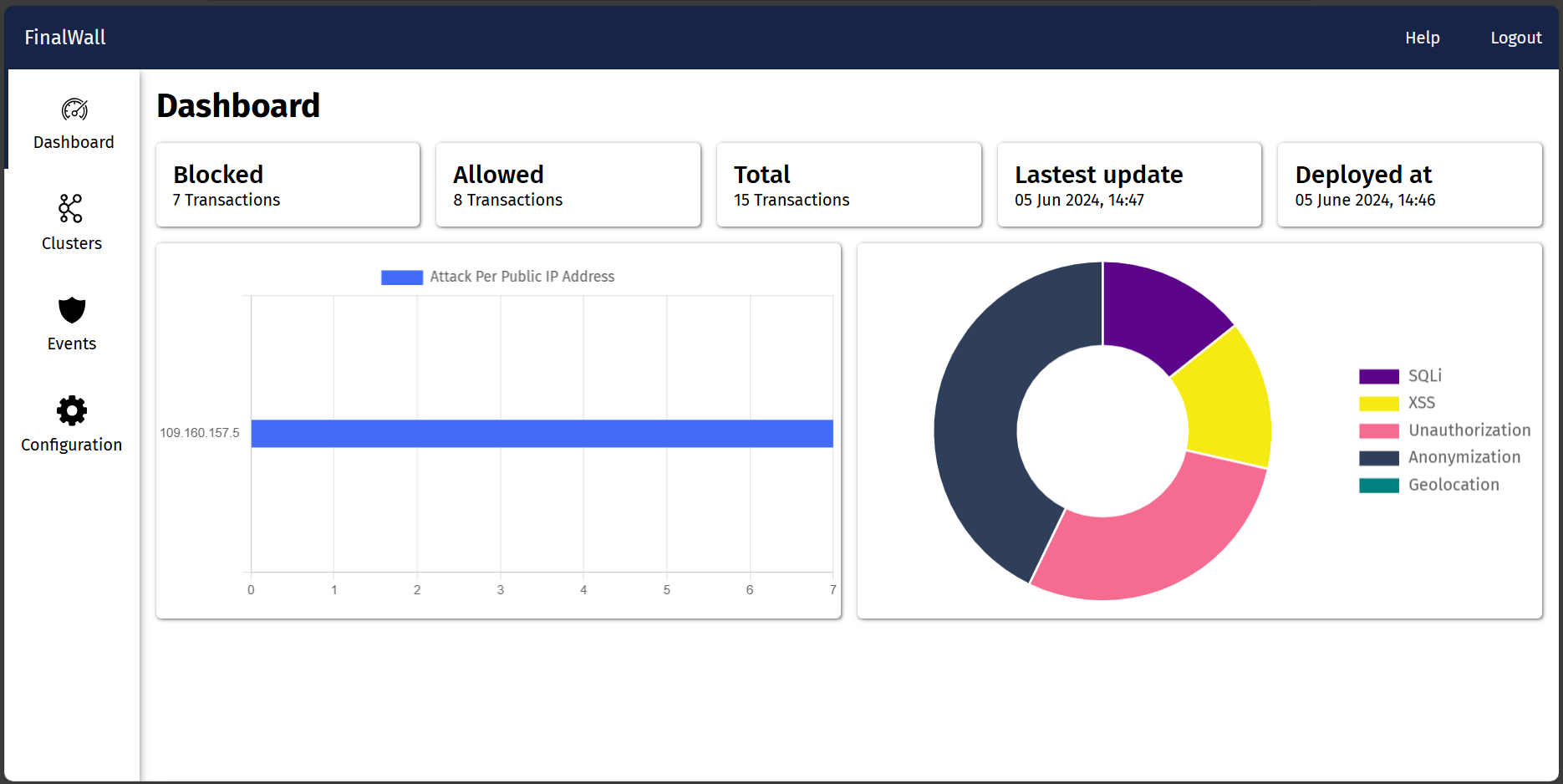
* FinalWall פותח על גבי פייתון גרסה 3.12 לכן יתמוך רק בגרסה זו.
* המערכת מזהה ניסיונות תקיפה על גבי מספר מוגבל של מקרים: SQLi, XSS, Broken Access Control, אנונימיזציה, מדינות עוינות ו-Source Blocking.
* המערכת הינה מביאה איתה מאגר חתימות ענק היות ומדובר בעולם ה-WEB, בכל יום מתקיימות כאלפי מתקפות סייבר וצוותי מחקר ברחבי העולם חוקרים אותן כדי למצוא חתימות חדשות למתקפות קיימות. לכן אפשרות של מאגר שמזהה את כל אופציות למתקפות בעולם הוא אפסי מאחר שיש אינספור גרסאות לשרשור מתקפות אלו. המערכת מביאה איתה מאגר חתימות מצומצם אך כזה שיודע לזהות Payloads רציניים של נוזקות כדי להראות שהוא יודע ליישם את הרעיון המחקרי. כמו כן, גם מוצרי WAF אמיתיים של חברות ענק אינם חוסמים את המתקפות כ-100% הצלחה. תמיד יהיו False Positives (הודעות מסוכנות אך עברו את המנגנון וקיבלו גישה לאתר) מכיוון שמדובר בעולם מאוד חברתי ומתמיד. האקרים ינסו 5000 קומבינציות של התקפות כדי להצליח להתקיף אתר אחד בודד. כמו כן, תמיד יהיו False Negatives (הודעות שהן בפועל חוקיות אך המערכת תקטלג אותן כמסוכנות). מאותה סיבה שציינתי מקודם.
  1. ממשקי פיתוח

כדי להבטיח מנגנון הגנה חזק ושימוש מיהמן של רשימות ACL המערכת משתמשת במספר APIs שונים אשר האמינות שלהם היא מירבית. המערכת משתמשת ב-3 ממשקים עיקריים:

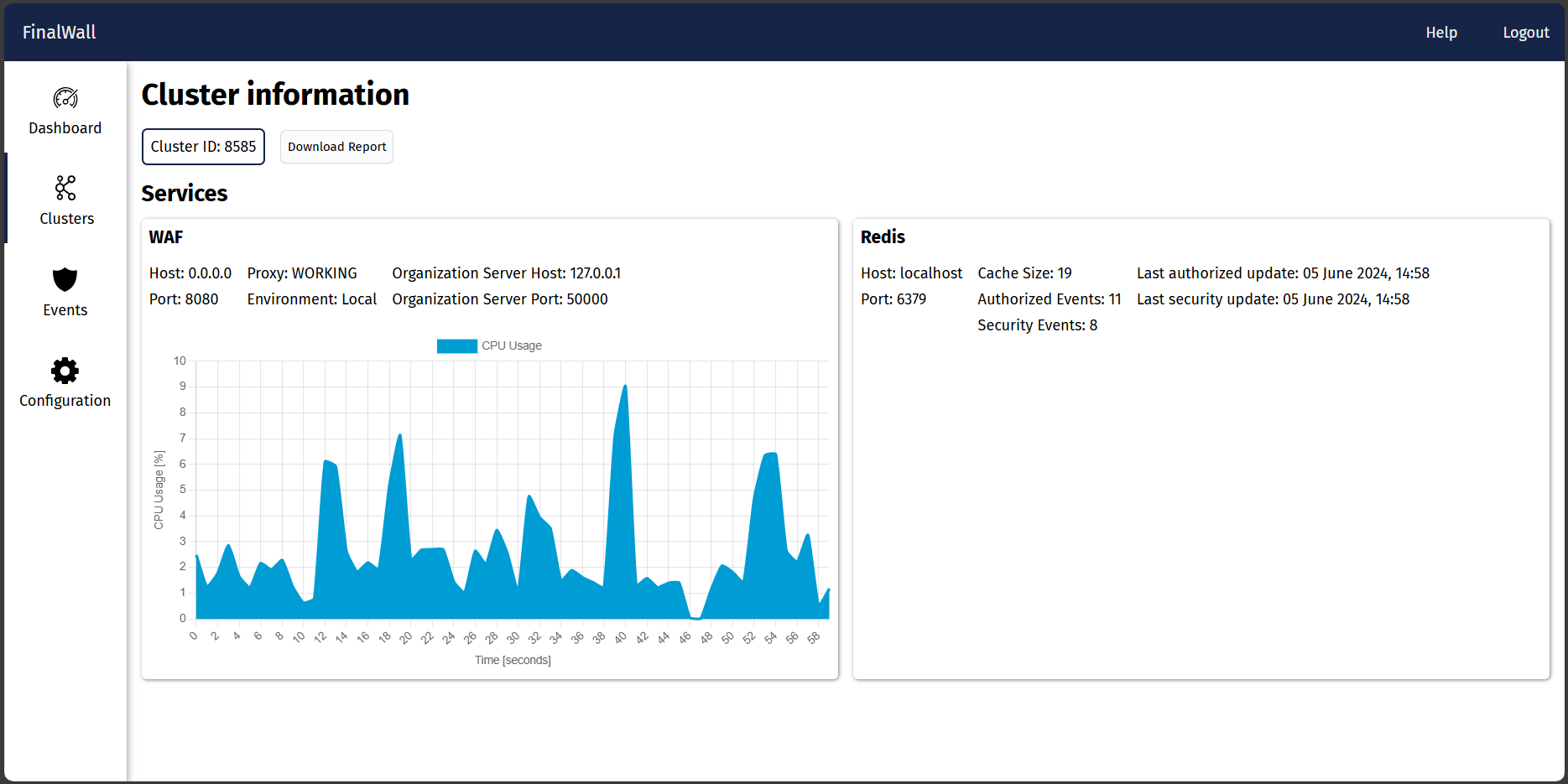
* + [Tor Exit Nodes API](https://check.torproject.org/exit-addresses) – ממשק למפתחים ולאנשים המזוהים בקהילה של Tor אשר מייבא רשימה של כ-1000 עד 2000 כתובות IP של exit relays the Tor. ה-ACL ששומר את הכתובות האלה מתעדכן כל חצי שעה מכיוון שבכל רגע יכולים ליצור עוד Relay שהוא Exit Node.
  + [MaxMind DB](https://www.maxmind.com/en/home) – API אשר שומר במסד נתונים נרחב מאוד את המיקום הגיאוגרפי של כל כתובת ציבורית בעולם. הוא שומר יבשות, מדינות ואפילו ערים עם דיוק של 99%. בכדי להעצים את היעילות של ACL ה-Geolocation המערכת משתמשת במסד נתונים אשר ניתן להורדה מהאתר של MaxMind וכך ניתן לקרוא ממנו ישירות מבלי להשתמש ב-API הוובי שלהם. המערכת משתמשת במסד הנתונים הידני בגלל שישנם ממשקים אשר מגבילים את כמות הבקשות ליום לכן מסד נתונים מובנה הינו פתרון טוב לבעיה זו.
  + Ipify – ממשק אשר יודע לייבא את הכתובת הציבורית של הנתב ברשת. ממשק זה משומש לצורך אירועים ולוגים הנובעים מתעבורה של Loopback ב-WAF.
  1. התייחסות לנושא אבטחה

המערכת היא פתרון אבטחה בפני עצמו אך מלבד להגן על מתקפות ב-WEB היא יודעת לזהות לקוחות אשר לא ניתן לסמוך עליהם. אלו לקוחות אשר נמצאים ב-ACL של אנונימיזציה ו-Geolocation. מכיוון שמדובר במדינות עוינות, או כלים לאנונימיזציה המערכת אינה יכולה לסמוך על תעבורה אשר באה משם והיא תחסום את הלקוחות. מעל שכבת הגנה זו מצטרפת שכבת הגנה נוספת יחודית בפרויקט והיא Source Blocking. יכולת יעודית לזהות ניסיונות התקפה חוזרים על עצמם מאותן כתובות. האקרים מנוסים לרוב ישלחו יותר מהודעה אחת ל-WAF לכן מנגנון זה פותח.

* 1. ממשק משתמש

ממשק המשתמש מבוסס על Framework של JavaScript הנקראת Vue.js. הממשק עצמו עולה ב-WEB וניתן לגשת אליו עם כל Browser שבידי המשתמש. **החלון הראשי - Dashboard:**  
  
התמונה להלן מתארת את ה-Dashboard לאחר כמה מתקפות על השרת. החלון הראשי – Dashboard מציע מגוון אפשרויות: צפייה בסטטיסטיקות על המערכת – מימין גרף Doughnut אשר מציג את סה"כ מתקפות שה-WAF בלם ואת הערך המספרי שלהן. כמו כן ניתן לפלטר מתקפות על ידי לחיצה על השם שלהן. מצד שמאל גרף עמודות לרוחב אשר מציג את חמשת כתובות ה-IP שהתקיפו את המערכת ואת מס' הפעמים שהתקיפו אותה.

מעל הגרפים מוצגות כרטיסיות עם מידע כללי על המערכת: כמה חסימות FinalWall בצעה, לכמה הודעות היא נתנה הרשאה, סה"כ הודעות עד כה, העדכון האחרון של המערכת ואת זמן הטעינה שלה.

**** **חלון ה-Cluster:** חלון זה מתאר את תמונת המצב הכללית של המערכת ואת הקבועים שלה. מצד ימין ניתן לראות מידע על מסד הנתונים Redis ועל כל ה- Caches שלו. מצד שמאל ניתן לראות את המידע הכללי על שרת ה-WAF. כמו כן, מצורף גרף CPU של שרת ה-WAF אשר מתעדכן בצורה אוטומטית מדי כל שניה.גם כאן, המידע מתעדכן בצורה אוטומטית ואין צורך ללחוץ כפתור ה-Refresh.

**חלון ה-Events:**

חלון זה מחולק ל-2 חלקים: רשימת הלוגים אשר מתארים טרנזקציות חוקיות, ורשימת

A screenshot of a computer

Description automatically generated הלוגים המתארים טרנזקציות לא חוקיות. להלן דף ה-Authorized Events (חוקיים).

**A screenshot of a computer

Description automatically generated אפשרות שניה – Security Events:**

ניתן לעבור בין חלונות ה-Events על ידי לחיצת הכפתור המתאים.

לחלונות אלו זה יש 3 חלקים מרכזיים: כפתורי המעבר בין Authorized ל-Security.  
 מנגנון הפילטר אשר מציג את האירועים אשר עונים על פילטר מסוים וטבלת האירועים אשר מתעדכנת באופן אוטומטי ב-RealTime.

* 1. תרחישים עיקריים

**A screenshot of a video game

Description automatically generatedהעלת שרת WAF:**

כאשר הWAF עולה לאוויר מתרחשים מספר תרחישים:

* + **Thread של ACL האנונימיזציה:**  
    A diagram of a computer

    Description automatically generated
  + **A black background with white text

    Description automatically generatedThread של Health Monitoring:**
  + **A diagram of a software application

    Description automatically generated with medium confidenceבזמן שכל זה קורה, הTunnel- מתחבר ל-API ושומר את כל ה-EVENTS בתור:**

**העלת שרת ה-FinalWallAPI:**  
A diagram of a web browser

Description automatically generated with medium confidence  
  
**העלת שרת ה-Frontend:**

A diagram of a process

Description automatically generated

**תרחישים אשר קורים במהלך פעילות ה-WAF:**

1. **זיהוי בקשת התחברות**

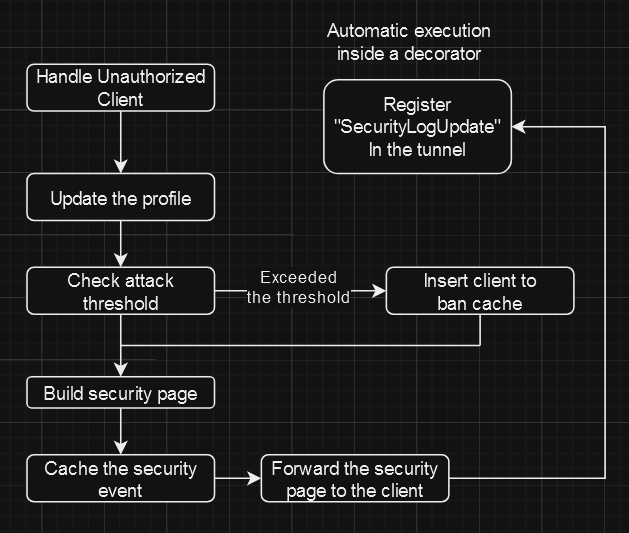
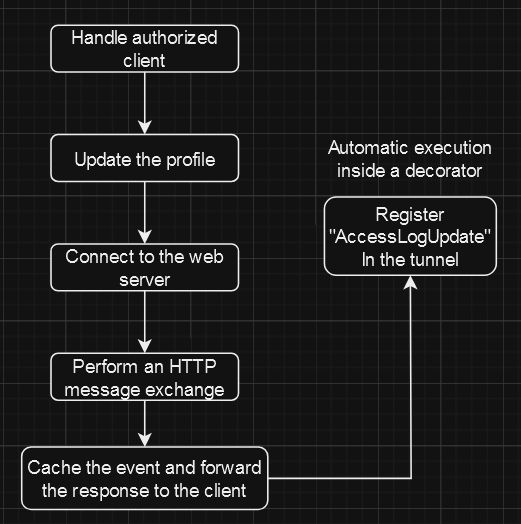
**A diagram of a system

Description automatically generated**

1. **בדיקת טרנזציה:  
   A black screen with white text

   Description automatically generated**

לאחר כל הבדיקות יוצרים לוג מתאים ו-EVENT מתאים במערכת ובמידה והבדיקות שליליות הפונקציה handle\_unauthorized\_client נקראת. אחרת, handle\_authorized\_client נקראת. הנה תרשמי הזרימה של אותן פונקציות כדלקמן:

1. **Handle\_Unauthorized\_Client:**
2. **Handle\_Authorized\_Client:**
3. תהליך כתיבת הפרויקט
   1. תהליך הפרויקט

בסוף כיתה י"א כבר אז התחלתי לחשוב על פרויקט וניסיתי להבין מה הנושא שאני הכי אוהב. אחרי הרבה זמן של חשיבה ועוד הרבה פרויקטים קטנים הגעתי להחלטה ליישם חומת אש אפליקטיבית. בהתחלה חששתי שלא אספיק ואני יכול לומר בוודאות שהצבתי לעצמי ציפיות יחסית גבוהות אך אני חושב שעמדתי בהן. כדי להתחיל בפרויקט הייתי צריך לחקור את עולם ה-WAF ולהבין איך שרתי Proxy עובדים, כיצד הם מקונפגים ואיך הם מתנהגים. התחלתי בלכתוב POC של פרוקסי פשוט ולאחר שבדקתי שהוא עובד סיימתי את אבן הדרך הראשונה והיא פיתוח שרת פרוקסי עם קונפיגורציה גמישה. יצרתי את אבן הבניין של שרת ה-WAF שלי. לאחר מכן, כדי להשלים את אבן הדרך הראשונה עליי היה להוכיח כי התעבורה אכן עוברת דרכי וכי אני מעביר אותה בצורה שלמה וחלקה ללא איבוד מידע. לאחר שהצלחתי אתגר זה הבטחון עלה והמשכתי עלאה.   
  
לאחר שהוכחתי כי יש לי Reverse Proxy יעיל ניגשתי לחלק הקשה בפרויקט – לחקור על חתימות של מתקפות Injection. היו לי הרבה ימים שחיפשתי מילות מפתח כמו Cheat Sheat ו-Signatures של התקפות הזרקה ולאחר הרבה חיפושים הגעתי לאוסף יחסית מצומצם אך כזה שיודע להגן על מגוון רחב מאוד של payloads. חשוב לציין שאין מאגר בנוי מראש לדברים כאלה. בחברות אשר מפתחות מוצרי הבטחה יש צוותים שלמים שהמטרה שלהם היא לבנות חתימות מאפס על ידי מחקר. לאחר שאספתי חתימות החלטתי שאני מיישם את אחד מה-TOP 10 והוא Broken Access Control, הוא בהחלט הכי קל. מכיוון שהגעתי לחלק הבעיה – החלק של האג'נדה אני הבנתי שאני צריך לפתח פרוטוקול מוצר מכיוון שלכל מוצר יש מגבלות כאלו ואחרות. כמו כן, אין חברה אחת בעולם שיודעת להגן מכל חתימה אפשרית – זה פשוט לא קיים. האגנדה שלי בנוגע ל-Broken Access Control היא לטעון כי המוצר שלי יודע לחסום כל בקשה למשאב אשר מופיעה בו אחת מן החתימות המבוקשות (אני מדבר על חתימות של paths ולא זה הזרקה). דוגמה לחתימה היא /admin/conf. פיתחתי מנגנון שחוסם את ה-TOP 10 הזה יחסית בקלות.

לאחר הפסקה רבה החלטתי להמשיך בפרויקט והגעתי לשלב הבא – מתקפות הזרקה. היה בידי אוסף של חתימות איכותיות והחלטתי כי אני אתחיל מ-SQL Injection. לאחר הרבה מחשבה ותכנון אלגוריתם פיתחתי שיטה יחודית. חיפוש חתימות יחודיות וחיפוש חתימות משולבים. לאחר שביצעתי Testing על מגוון רחב של Bypass payloads עם Case Toggeling. (שיטת עמעום שמשנה את גודל האותיות מ-upper ל-lower ולהפיך – כדי להגביל חתימות regex). כאן התחלתי לצבור בטחון ושילבתי מנגנון הגנה מפני כל שיטות ה-Bypass שציינתי עד כה. לאחר שהצלחתי ליצור מנגנון הגנה חזק נגד SQL Injection. לאחר מכן עברתי על אותו תהליך ב-XSS שזה פשוט חיפוש חתימות פשוט בהודעה. בשלב הזה פיתחתי מנגנון Source Blocking פשוט וכל מה שנותר לי אחרי זה, זה ליצור ACLs שונים בלפחות 3 איזורים. בחרתי ב-ACL של טור, ACL גיאוגרפי ו-ACL של כתובות לא חוקיות או כתובות אשר חסומות במערכת. לאחר מכן, התחלתי לפתח תווך תקשורת בין ה-WAF לבין ה-API שבניתי ופיתחתי פרוטוקול מנהרה פשוט וקל, ולבסוף UI. במהלך הפרויקט הבנתי את הגודל של פיתוח פרויקט גדול שזה ובהחלט למדתי הרבה מן התהליך.

* 1. אתגרים ואופציות שונות למימוש

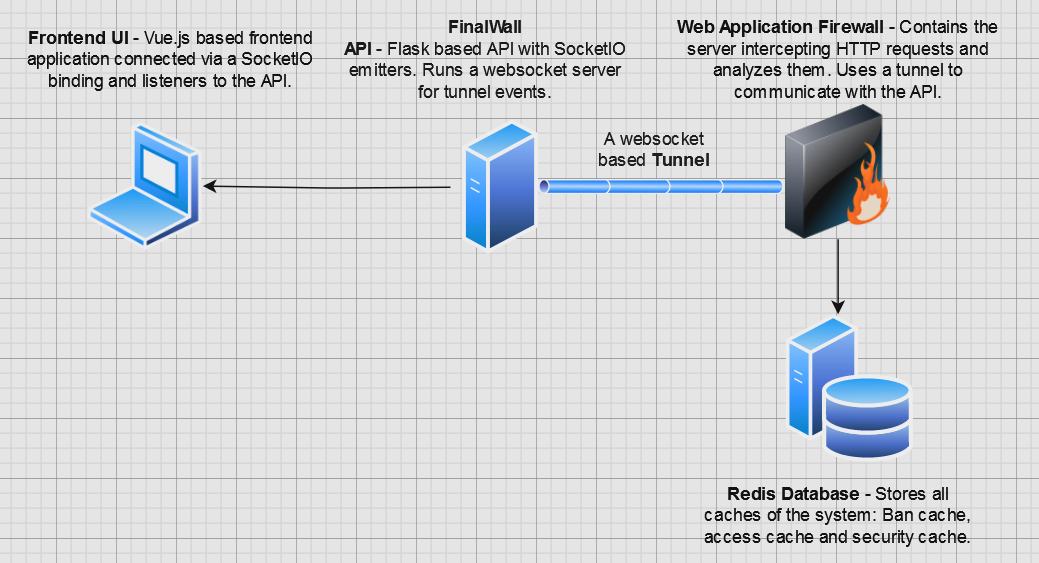
האתגרים שחוויתי במהלך פיתוח נעו בין חוסר מוטיבציה ופחד לסיים בזמן אך התגברתי עליהם ככול שהזמן עבר.

* האתגר הפיתוחי הראשון היה התלבטות בשפת תכנות – אני מעריץ גדול מאוד של ראסט ובאותה תקופה לא רציתי לפתח בפייתון אך עם הרבה מחשבות הבנתי שפייתון עדיפה. היא קלה יותר ומהירה יותר ב-Long run של פיתוח - (לא מבחינת מהירות, אלא מבחינת מהירות פיתוח).
* האתגר השני שלי היה לנסות לשלב בין שפות – רציתי ממשק פייתוני אך עם יכולות ניתוח מהירות. תחילה ניגשתי ל-Cython במטרה לכתוב חלק מן הפונקציונליות שלי בשפת C אך מהר מאוד הבנתי שלא כדי לי – לא בגלל קושי השפה אלא בגלל קושי בMigration עתידי. החלטתי שאני עוזב את הרעיון וכותב פרויקט שהוא Pure Python.
* האתגר השלישי הוא לבחור Framework ל-UI ובכלל, איך לעשות UI, עיצוב ופיתוח Frontend כללי. לאחר הרבה פרויקטים עם CustomTkinter הבנתי שאני יכול לפתח בפייתון יותר. אני צריך הפסקה ולכן בחרתי UI ב-WEB. היה אתגר חדש וזה ללמוד שפת תכנות חדשה שזה JavaScript ולהבין איך JS Frameworks עובדים. תכנות בשפת VUE הוא שונה מאוד מתכנות רגיל עם מודולים של UI בפייתון. אני שמח מאוד עם ההחלטה שלי לפתח UI בWEB כי למדתי המון על איך Real World Applications בנויים.

1. מרכיבי פתרון
   1. תיחום הפרויקט

* תקשורת – ובכן, כל הפרויקט שלי הוא תקשורת אחת גדולה! מדובר בשרת אסינכרוני מבוסס Streams אשר מטפל בהודעות HTTP אשר תוך שימוש ב- Concurrency ומנהרות מתקשר עם שרת API שהוא מבוסס Flask, SocketIO ו-WebSockets. כל התקשורת הינה רוכבת על פרוטוקול TCP ו-HTTP.
* מערכות הפעלה – ב-2 השרתים שצויינו לעיל ישנם מספרים Threads שעובדים במקביל.  
  כמו כן כל שרת הWAF הוא אסינכרוני. הוא כולל יישום ובנוי על תכנות אסינכרוני בשימוש של async, await. כמו כן הוא משלב בתוכו מגוון רעיונות כמו תור אסינכרוני, משימות – Tasks, רוטינות – Coroutines ו-Event loops.
* תצוגה – גרפיקה מושקעת וידידותית מבוססת Vue.js.
* אבטחה – הפרויקט כולו מממש מנגנון הבטחה רחב – סינון כתובות, מתקפות וניסיונות התקפה מתמידות.
* מבנה נתונים – מבני נתונים נרחבים הכוללים תורים, רשימות, Sets, מילונים.  
  במטרה להקל על שיתוף מידע ו-Events בין ה-API ל-WAF ובין ה-API לשרת Frontend.  
  כמו כן היות ויש מעבר שפות מה-API לשרת UI (פייתון לjs) השתמשתי גם ב-json וגם בPICKLE כדי להקל על שמירת נתונים.
* שימוש במסד נתונים – שימוש נרחב במסד הנתונים Redis אשר אפשר לי ליישם Caches בצורה יעילה, מהירה ומיטבית. שימוש ב-SQLITE כדי למפות פרופילים ללקוחות.
* ארכיטקטורת קוד – שימוש מרובה במחלקות מכמה סוגים: מחלקות רגילות ומחלקות dataclass. מבנה נכון וקל של פרויקט ומחולק לקבצים, תקיות מודולים ו- Packages.
* תיעוד – תיעוד המשכי בקוד עצמו תוך שימוש בdocstrings ושימוש ב-GIT ו-GITHUB בשביל פרסום ה-source code.
  1. סביבת העבודה (טכנולוגיה)
* **שפות התכנות – שפה ראשונה:** פייתון הינה השפה העיקרית של הפרויקט – כל הBackend בנויה עליה.
  + **שרת WAF:** לכתוב שרת אשר מסוגל לנהל תקשורת רציפה עם לקוחות, וניהול מנהרה ברקע. לשם כך פייתון מציעה מגוון ספריות builtin והעיקרית מבינהן היא asyncio.
  + **פיתוח API:** פייתון הינה חזקה מאוד בתחום התקשורת לכן ה-API בנויה עליה גם כן. דבר זה מקל על type casting בין שפות כאשר נוגע לתקשורת בין הWAF לAPI.
* **שפה שניה:** VueJS ו-JavaScript. כבר שנים שJS הינה שפה הבונה את המנועים של כל הFrontend בכל ממשק גרפי לכן בפיתוח WEB כדי מאוד לגשת ל-js. היום יש תחליפים טובים יותר אך JavaScript פשוטה ובחרתי ללכת על אופציה קלה ליישום. VUEJS הינה שפה בפני עצמה כשזה בא ל-templating של דפי האתר ושילוב האינטרקציה עם המשתמש. שפות אלו הביאו לתוצאות מעולות ב-UI.
* **סביבת פיתוח:** לשפת Python השתמשתי ב-Pycharm ו-Visual Studio Code ל-Vue.js ו-Js.
  1. מבט טופולוגי

זהו תרשים טופולוגיה של המערכת הכללית. מתוארת חומת אש אפליקטיבית אשר מחוברת במנהרת TCP ל-API במטרה לשלוח Events מכל לקוח שהתחבר (ראו תרשים הבא).  
לכל WAF יש Redis Database ששומר 3 Caches מרכזיים: Cache של חסימות, Cache של אירועים חוקיים ואירועים לא חוקיים. לאחר כל Event שמעודכן על ידי הWAF בAPI, הAPI עצמו יעשה forward ל-Event עם SocketIO emitter. (כמו send, רק של socketIO).



בתרשים זה מתוארת טופולוגיה כללית יותר מצד הלקוח.

A black square with a fire on it

Description automatically generated

* 1. מבנה נתונים

WAF:

* + Ucid – מזהה יחודי ל-WAF
  + משתנה מטיפוס WafConfig אשר שומר את הקונפיגורציה של ה-WAF.
  + Local – משתנה בוליאני אשר מתאר את רשת ה-WAF (LAN בלוקלי או הקשבה ב-WAN בRemote).
  + Server – שרת של asyncio מטיפוס Server.
  + Deploy\_time – תאריך (יום\חודש\שנה, שעה:דקות) של זמן ה-run של ה-WAF.
  + Health – תור מסוג Deque. עוטף "רשימה" השומרת את ערכי צריכת ה-CPU של המערכת בדקה האחרונה.
  + With\_tunneling – משתנה בוליאני אשר אומר ל-WAF האם ליצור מנהרה ל-API או לא.
  + BanManager – מנהל Cache של Redis האחראי על Cache החסימות.
  + EventManager – מנהל Cache של Redis האחראי על 2 Caches – Security ו-Access events.
  + ProfileManager – מנהל פרופילים. אחראי על מסד נתונים מסוג SQL הממפה פרופיל לכל hash של לקוח.
  + ACL – Access Control List הנוגע לאנונימיזציה. משתנה מסוג Acl שעוטף רשימה של כתובות IP.
  + Address – tuple ששומר את הכתובת של ה-WAF
  + Target – tuple ששומר את הכתובת של השרת שעליו ה-WAF הופקד.
  + State – מצב ה-WAF. המצבים הם כדלקמן: Working, Deployed, Closed, Created.

Transaction:

* + Owner – טיפוס מסוג HostAddress (דמוי tuple). שומר את הכתובת של בעל ההודעה.
  + Real\_host\_address – הכתובת האמיתית (שאפשר לסמוך עליה) של בעל ההודעה.
  + Raw – משתנה מסוג bytes. מחזיק את הודעת ה-HTTP במלואה.
  + Creation\_date – מחרוזת שמסמלת את תאריך זיהוי ההודעה במערכת.
  + Method – משתנה מסוג bytes או מחרוזת. מצביע על סוג המתודה של ההודעה.
  + url – משתנה מסוג ParseResultBytes – מחזיק את הURL של ההודעה.
  + Version – משתנה מסוג bytes או מחרוזת. מצביע על סוג הפרוטוקול של HTTP שדרכו ההודעה עברה.
  + Query\_params – משתנה מסוג מילון אשר ממפה כל ערך (query) ב-URL לערכים שלו.
  + Headers – משתנה מסוג מילון אשר ממפה כל Header לערך שלו.
  + Body – משתנה מסוג bytes או str. מחזיק את ה-raw bytes של החלק של ה-body. (החלק לאחר ה-\r\n\r\n).
  + Size – משתנה מטיפוס int. מצביע על גודל המחרוזת.

AccessLog:

* + Ip – משתנה מסוג מחרוזת. מצביע על הIP של בעל ההודעה שעשתה trigger לאירוע.
  + Port – משתנה מטיפוס int. מספר הפורט שממנו ההודעה נשלחה.
  + Download – משתנה בוליאני אשר מצביע האם ניתן להוריד log מסוים כקובץ log למחשב.
  + Sys\_epoch\_time – משתנה מסוג float. מחזיק את כמות הזמן שעבר Since the Epoch.
  + Creation\_date – מחרוזת שמצביעה על תאריך קליטת ה-Transaction.
  + Geolocation – משתנה מטיפוס GeoData (ראה מבנה הבא).

GeoData:

* + Continent – יבשה.
  + Country – מדינה.
  + City – עיר.

Security Log:  
הערה – אותו מבנה בדיוק רק עם תוספות למבנה של AccessLog.

* + Classifiers – רשימה של classifiers של כל לוג.
  + Malicious\_data – המידע הרגיש שגרם לאותו אירוע לקבל Trigger במערכת. במיוחד במתקפות.
  + Metadata – משתנה מסוג מילון המחזיק מטא נתונים על אותו לוג לפי מידת הצורך.

Event:

* + Kind – משתנה מסוג int. מצביע על סוג האירוע. (AUTHORIZED או UNAUTHORIZED, או CONNECTED)
  + Id - תלוי לפי סוג האירוע. יכול להיות קוד ארוך של HASH או קוד בעל 8 תווים שמזוהה עם מתקפה.
  + Request – משתנה מסוג Transaction. הבקשה של הלקוח לשרת.
  + Response – משתנה מסוג Transaction. התשובה של השרת לבקשת הלקוח.

Profile:

* + Host – מחרוזת ששומרת את ה-IP של אותו פרופיל.
  + Connection\_date – מתי הוא התחבר לראשונה.
  + Last\_used\_port – הפורט האחרון שהשתמש.
  + Last\_connection\_time – זמן ההתחברות האחרון שלו.
  + Last\_event – משתנה מטיפוס Event. מיוחס לאירוע האחרון שאותו לקוח ביצע.
  + Attempted\_attacks – מספר הפעמים הוא התקיף את השרת.
  + Last\_attempted\_attack – סוג המתקפה האחרונה שהוא עשה.

SignatureDB:

* + Sql\_data\_set – משתנה מילון הממפה מילה לכל רמת חשיבות שלה לאלגוריתם של SQL Injection.
  + Xss\_data\_set – מאגר חתימות של xss מסוג מילון.
  + Broken access control set – מאגר של חתימות של משאבים אסורים לאף אחד.
  1. מסד נתונים

המערכת משתמשת ב-2 מסדי נתונים. העיקרי מבניהם הוא Redis.

1. BanManager – לקוח Redis אשר מנהל את ACL החסימות אשר בנוי על מנגנוןSource Blocking. שומר כל Key כ-bans:{hash} כאשר כל Key מייצג Hash של לקוח (כתובת ה-IP) שלו. Manager זה ממפה לכל Key 2 ערכים: banned\_at, duration\_remaining. הManager משתמש בפונקצית EXPIRE של רדיס ומוחקת את ה-Key מהמסד נתונים לאחר שזמן החסימה עבר. המערכת תזהה אם הכתובת חסומה רק אם היא מופיע במסד הנתונים תחת namespace של bans.
2. EventManager – לקוח Redis אשר מנהל 2 namespaces שהם Access Events, וSecurity Events. המערכת מיישמת Cache לשני סוגי ה-Events אשר משתמש ב-Ordered Set. ב-Access Events נשמרים אובייקטים מסוג Event שה-kind שלו הוא authorized בצורה Pickle. כמו כן, Cache של Security Events שומר אובייקטים מסוג Event עם kind של Unauthozied\_request. המערכת תוסיף ל-Cache את ה-Events לאחר סיום חקירת ההודעה.
3. Profiler – מסד נתונים מבוסס SQLite אשר שומר לכל Hash של לקוח (hash של כתובת ה-ip שלו) את הפרופיל שלו. הפרופיל שמור בצורת Pickle.
   1. מבט מודולרי

החלקים המרכזיים בפרויקט:

* + AsyncStream – משתנה שעוטף Reader ו-Writer של Streams של Asyncio. הוא ה-underling component בתהליכי שיתוף מידע בין ה-WAF לשרת, בין ה-WAF ללקוח.
  + Connection – משתנה המייצג חיבור לנקודה מסוימת ברשת. עוטף את AsyncStream. ממנו שולחים ומקבלים מידע – אלגוריתמים אשר מבוצעים על ה-AsyncStream.
  + Transaction – טיפוס אשר מייצג הודעת HTTP משני הצדדים אשר כולל בתוכו חלקים מפורטים על כל חלק בהודעה.
  + Log – Log הינו טיפוס אב ל- Access Log ו-Security Log שהם 2 סוגים Logים בפרויקט. מייצגים פירוט נרחב יותר על האירוע.
  + WAF – הטיפוס העיקרי בפרויקט. אחראי על טיפול בהודעות אשר עוברות דרכו ודיווחן ל-API בצורה דינמית.
  + API – אחראי על שמירת Events והעברתם לשרת ה-Frontend. מנהל תקשורת רציפה עם ה-WAF על ידי ניהול שרת Websockets.
  + ACL – מסווג ל3 סוגים אשר אחראים על פילטור התעבורה.
  + Tunnel – מנהרת תקשורת של ה-WAF עם השרת ה-API. שולח Events לשרת וכך גורם לעדכון UI בזמן אמת.
  + TunnelEvent – Enum המכיל את כל סוגי האירועים שניתן לעדכן במנהרה.
  + Check – טיפוס אשר מכיל בדיקת שיש לעשות ואת הtransaction שעליה לבצע את הבדיקה.
  + ToJson – המערכת משתמשת בחלקה היודעת לעשות serialize בין Pickle ל-json כדי להעביר את המידע השמור במסדי הנתונים לfrontend. (js מבין רק json). טיפוס זה הוא טיפוס גלובלי ב-API הקרוי JSONIZER.
  1. פירוט מודלים עיקריים

|  |  |  |  |
| --- | --- | --- | --- |
| Description | Input\Output | Function | Class |
| יוצרת שרת אסינכרוני והופכת את מצבו של הWAF ל-Deployed. עטופה בדקורטור אשר שולח TunnelEvent מסוג WafServiceUpdate. | Input- output- | Deploy | Waf |
| הופכת את מצבו של הWAF ל-Working, יוצרת 2 Threads – ACL ו-CPU ומתחילה 2 משימות ברקע: ניסיון התחברות לAPI על ידי מנהרה ותחילת שרת הWAF המרכזי). | Input- output- | Work |  |
| מטפלת בהתחברות לקוח. יוצר Connection ומתחיל את סדרת הבדיקות על הלקוח. תחילה מבדיקת לקוח חסום בACL ה-Source Blocking. לאחר מכן בדיקת לקוח מלוכלך (אנונימי או ממדינה עוינת) -> בדיקת ההודעה ששלח (check\_transaction). אם בדיקת מסוימת כשלה הפעולה נעצרת ועוברת ל-handle\_unauthorized\_client אחרת מטפלת בו כלקוח חוקי. עטופה בדקורטור אשר שולח TunnelEvent מסוג WafServiceUpdate. | Input- reader, writer output- | Handle\_connection |  |
| מעדכנת forensics על הלקוח בונה לו security page ומעבירה לו אותו. בשלב זה נסיון החיבור שלו לשרת לא צלח וההודעה נזרקת. עטופה בדקורטור ששולח TunnelEvent מסוג WafSecurityLogUpdate | Input-client, event output- | Handle unauthorized\_client |  |
| מטפלת בנסיון חוקי של הלקוח לשימוש באתר. מעדכנת פרטים על הלקוח ומעבירה לו את התשובה מהשרת. עטופה בדקורטור ששולח ל-API TunnelEvent מסוג WafAccessLogUpdate. | Input-client, event output- | Handle\_authorized\_client |  |
| מנסה להתחבר לendpoint הנקבע בטיפוס בעת יצירתו. לאחר שהtimeout נסיון התחברות זה מפסיק. שומרת את כל ה-Events שנשלחים דרך ה-Tunnel בתור ולאחר ההתחברות המנהרה שולחת הכל in order. | Input-timeout output- | connect | Tunnel |
| שולחת Event בתוך המנהרה. | Input-event output- | Register\_event |  |
| פותחת נתיב תקשורת דרך aynscio עם המחשב המייצג את (ip, port). | Input-ip, port output-AsyncStream | Open\_stream | AsyncStream |
| מחזירה את הטיפוס עצמו (שימוש לasync for) פונקציית קסם בפייתון. (builtin). | Input- output- | \_\_aiter\_\_ |  |
| באופן אסינכרוני קוראת מהstream שנוצר ומחזירה את המידע שנשלח. | Input- output-bytes | \_\_anext\_\_ |  |
| פונקציה חכמה אשר קוראת מידע מ-asyncstream בצורה אסינכרונית עד שcondition קורה. | Input-condition, args output-bytes | Recv\_until | Connection |
| קוראת לכמה פונקציות פנימיות במחלקה אשר יודעים לפרק מתוך ה-raw bytes של ההודעה (HTTP) את כל חלקי ההודעה – מתודה, URL, פרמטרים, גוף ההודעה, headers וכו. | Input- output- | process | Transaction |
| כאשר מבצעים בדיקה מבצעים אותה על ידי check.fn() (ניגשים לפונקציה שהcheck שומרת( |  | שומר בתוכו 2 שדות: בדיקה וargs לבדיקה. | Check |
| מבצעת המרה בין צורת Pickle לצורת json. | Input-event or transaction output-serialized\_event,serialized\_transaction | \_\_to\_json | JSONIZER |
| מעדכנת את הרשימה של כתובות אנונימיזציה כל חצי שעה. | Input- output- | Activity loop | ACL of anonymization |
| עושה serialize לjson תוך שימוש בJSONIZER. תלוי event\_type. | Input-event\_type, data output-proccesed data | Process\_event\_data | API |

1. תסריטי בדיקה
   1. דגשים בבדיקה  
      1) המערכת יודעת להתחבר לשרת API ולשמור Events בתור.  
      2) המערכת מקבלת הודעת HTTP, יודעת ליצור transcation ממנה.  
      3) המערכת יודעת ליצור בדיקות ולבצע אותן על כל טרנזקציה.  
      4) המערכת יודעת לסווג כתובת IP לחוקית ולא חוקית על פני 3 ACL שונים.  
      5) Register\_Event עובד ומעדכן את שרת הAPI בכל עדכון.  
      6) עדכון תקיף של כל Cache ושל כל ACL.
   2. תסריטי בדיקה עיקריים  
      1) תהליך deployment של WAF בהצלחה.  
      2) תהליך Working של ה-WAF אשר מאתחל 2 Threads ו2 משימות ברקע – CPU, ACL ומשימות של חיבור מנהרה והקשבה אינסופית לחיבורים. (שרת).  
      3) הרמת שרת API לפני הרמת הWAF במעבדה תקינה.  
      4) הרמת שרת WAF לפני הרמת API במעבדה פחות תקינה – (מקרה זה פשוט שומר את ה-Events בתור לעומת תסריט מספר 3 שיותר אידיאלי).  
      5) בדיקת אנונימיזציה על ידי ACL מבוסס TOR.  
      6) בדיקת SQLi, Broken Access Control ו-XSS. על ידי שימוש ב-CRS מובנה.  
      7) בדיקת Dirty Client על ידי בדיקה גיאוגרפית.  
      8) עדכון אוטומטי של המערכת ב-UI.

1. רפלקציה
   1. לוח זמנים מוערך לניהול הפרויקט:

|  |  |
| --- | --- |
| תקופה בחודשים | התקדמות |
| נובמבר | התחלת מחקר בנושא WAF והבנת המוצר. |
| נובמבר - דצמבר | בניית POC הכולל Reverse Proxy שיודע להעביר לקבל מידע בצורה אסינכרונית. |
| ינואר | חקירת עולם הWEB במטרה למצוא חתימות של הזרקה ושל Broken Access Control. |
| פברואר – תחילת מרץ | סיום פיתוח לוגיקה אשר חוסמת מתקפות – SQLi, XSS, ו-Broken Access Control. |
| מרץ | צירוף מנגנוני חסימה מתקדמים ומתמידים – ACL ו-Source Blocking. |
| שבוע ראשון של אפריל | כתיבת API אשר מקבל Events מ-Tunnel |
| המשך אפריל - מאי | כתיבת שרת Frontend עם ממשק משתמש קל וידידותי ושילוב הBackend עם הFrontend כדי להציג סטטיסטיקות על הפרויקט בצורה מיטבית ו-Real Time. |
| המשך מאי ותחילת יוני | סיום הפרויקט עם תיקונים קטנים וספר פרויקט. |

* 1. אתגרים ותרומה אישית  
     הפרויקט הוא התוצר הגדול ביותר שפיתחתי. אני אוהב לפתח ולראות את הדבר הזה יוצא לפועל בהחלט תרם לי הרבה בתפיסת העולם כשזה בא לניהול פרויקטים גדולים. שקדתי עליו תקופה מאוד ערוכה ואני בהחלט יכול לומר כי אני השגתי את המטרות שלי ואני שמח מאוד עם התוצר הסופי שלי. בהתחלה היה לי קשה להאמין אבל גם ידעתי שאצליח.   
       
     כאחד שאוהב את עולם ה-WEB ולא עשה פרויקט אחד הקשור אליו, לקחתי על עצמי אתגר מאוד גדול וכשהתחלתי אותו כבר הבנתי לאן אני נכנס. מדובר בעולם עצום שמתפתח כל רגע והיו לי רגעים ששאלתי את עצמי – אבל מה לעשות אם א' קורה? בגלל ש-ב' עלול לקרות. ככה הכנסתי את עצמי ללופ של שאלות עם עצמי ועל המשך הפיתוח הפרויקט.  
     היו לי ימים שבהם גם לא עבדתי עליו אבל כן חשבתי כיצד אני ממשיך הלאה. עם הזמן השתפרתי בפיתוח והבנתי ניואנסים קטנים אך כל כך חשובים שמפרידים מפתח טוב ממפתח מדהים. פיצול תחום העבודה לאבני דרך בהחלט עזר לי בפיתוח הפרויקט. אני רוצה לציין כי ליצור פרויקט כזה גדול הוא לא מובן מאליו בשבילי מכיוון שאני הצלחתי ליצור מוצר שמתחרה עם השוק העולמי ובשבילי זוהי ההצלחה. למדתי הרבה מאוד במהלך הפרויקט ויש לציין שאפילו התבגרתי! למדתי שיעורים חשובים לחיים. בכל זאת – תקופה של יותר מחצי שנה של פרויקט רציני.
  2. תובנות  
     היו לי הרבה רגעים של ייאוש ומבט על מסך המחשב. היו לי ימים שבהן בהיתי במסך ולא קרה כלום. ברגעים כאלה הבנתי שהדבר החשוב ביותר הוא מוטיבציה ואהבה עצמית – הרי אני עברתי דרך ארוכה, הקדשתי זמן לדבר הזה ולהפסיק את זה באמצע זה כמו להשליך את הפרויקט לפח.  
       
     ניהול זמן – הבנתי איך לנהל את הזמן שלי. ארגנתי את סביבת העבודה שלי, אבני דרך, סדרי עדיפויות. דברים אלו פתחו אצלי משמעת ואופן פיתוח רציף בפרויקט. עם לחץ הזמן אני יכול לומר שהתפתחתי באופן פתרון הבעיות שלי – הייתי צריך להביא פתרון יצירתי אך נכון לבעיה מסוימת והצלחתי.   
       
     פרויקט זה דרש מסירות, התמדה, רצון ויכולת ואני גאה בו אינספור פעמים. למדתי בו המון ופיתחתי הרבה מאוד יכולות אשר ילוו אותי בפרויקטים עתידיים ובכלל – בהמשך החיים.

1. הוראות התקנה ותפעול
   1. תצורה ודרישות קדם  
      **לשרת ה-WAF:**   
      נדרש Python Interpreter של 3.12+. כמו כן המערכת הינה Cross Platform לכן כל מערכת הפעלה יכולה לעבוד. להלן רשימת הdependencies שיש להתקין.  
      ,psutil~=5.9.6 ,websockets~=12.0 ,Jinja2~=3.1.2 ,redis~=5.0.4 ,pytz~=2023.3.post1 DateTime~=5.3 ,requests~=2.31.0 ,geoip2~=4.8.0.  
      בשביל להצליח להשתמש ב-Redis צריך להוריד את אותו למחשב. נע לעקוב אחר המאמר הזה:  
      <https://redis.io/docs/latest/operate/oss_and_stack/install/install-redis/>   
        
      **לשרת FinalWall API:** נדרש Python Interpreter של 3.12+. רשימת dependencies:  
      Flask~=3.0.0, websockets~=12.0.  
        
      **לשרת ה-Frontend של Vite.**  
      Node.js גרסה 20+. וגרסה אחרונה של vue-js.
   2. התקנה  
      **הוראות התקנה לשרת WAF:**  
      לאחר התקנה כל ספרייה ותוכנות, יש להריץ את שרת הרדיס המקומי במחשב. הפקודות הינן זהות בWindows ובלינוקס רק שבווינדוס יש להריץ את Redis על גבי WSL.

sudo service redis-server start

לאחר הרצת מסד הנתונים יש להיכנס לתקיית ה-Root של הפרויקט, לשנות את הקונפיגורציה כרצונכם ולוודא שהאתר תחת הLabel – WebServer הוא הנכון והוא פועל. אם אין אתר ניתן למצוא אתר לדוגמה על ידי הרצת תוכנת bWAPP. אם ישנה העדפה לאתר קל ופשוט שניתן להרים ברגע יש לעקוב אחר המדריך הבא – אתר לדוגמה אשר פותח על ידי למען הפרויקט: <https://github.com/dayeya/SQLi>.   
שם ניתן להוריד את האתר מבוסס Flask ו-Sqlite ולהריץ את הפקודה הבאה בתקיית הפרויקט:

python main.py

לאחר וידוא הכתובת של שרת הWEB ושאר קבועי קונפיגורציה יש להריץ בתקיית הפרויקט הראשי את הפקודה הבאה:

python finalwall.py

**הוראות התקנה לשרת API:**לאחר הורדת כל הספריות והתוכנות יש לגשת לתקיית הbackend בפרויקט שם תמצאו את שרת ה-API. יש להריץ את הפקודה:

python fwserver.py

**הוראות התקנה לשרת ה-UI:**לאחר התקנת Node.js וVUE על המחשב יש להיכנס לתקייה של web/frontend ולוודא כי הכתובת של API הקובץ main.js הינה הנכונה, לאחר מכן יש להריץ:

npm install dev  
npm run dev

1. ביבליוגרפיה

במהלך כתיבת הפרויקט הסתמכתי על מספר מקורות מידע:  
<https://portswigger.net/web-security/sql-injection/cheat-sheet>  
<https://portswigger.net/web-security/cross-site-scripting/cheat-sheet>

<https://owasp.org/www-project-top-ten/>

<https://owasp.org/>

<http://projects.webappsec.org/w/page/13246985/Web%20Application%20Firewall%20Evaluation%20Criteria>

<https://www.getsecureworld.com/blog/the-most-powerful-waf-evasion-techniques/>

<https://www.scaler.com/topics/cyber-security/web-application-firewall-bypass-techniques/>

<https://owasp.org/Top10/A01_2021-Broken_Access_Control/>

<https://support.radware.com/app/answers/answer_view/a_id/15504/~/appwall-source-blocking-feature-algorithm>

<https://www.maxmind.com/en/home>

1. נספחים

את הפרויקט ניתן למצוא ב-Github: <https://github.com/dayeya/FinalWall>  
  
להלן נספחי הקוד:

class Waf:  
 *"""  
 A class representing a Web application firewall.  
 Protects a \*single\* entity in the network.  
 """* def \_\_init\_\_(  
 self,  
 conf: WafConfig,  
 ucid: int,  
 local: bool=False,  
 with\_tunneling: bool=False,  
 ) -> None:  
 try:  
 SignatureDb()  
 except Exception as \_database\_loading\_err:  
 print("ERROR: could not initialize database due:", \_database\_loading\_err)  
  
 self.\_\_ucid = ucid  
 self.\_\_config = conf  
 self.\_\_local = local  
 self.\_\_server = None  
 self.\_\_deploy\_time = None  
 self.\_\_health = deque([])  
 self.\_\_with\_tunneling = with\_tunneling  
 self.\_\_tunnel = Tunnel(conf.admin["backend\_api"])  
 self.\_\_ban\_manager = BanManager()  
 self.\_\_event\_manager = EventManager()  
 self.\_\_profile\_manager = ProfileManager()  
 self.\_\_acl = AccessList(main\_list=[],  
 api=conf.acl["api"],  
 interval=conf.acl["interval"],  
 backup=conf.acl["backup"])  
 self.\_\_address = (conf.waf["ip"], conf.waf["port"])  
 self.\_\_target = (conf.webserver["ip"], conf.webserver["port"])  
 self.\_\_state = \_WafState.CREATED  
 print(f"INFO: Waf created")  
  
 async def \_\_cpu\_loop(self):  
 *"""Sends a CPU usage every second."""* while True:  
 self.\_\_health.append(self.get\_health())  
 if len(self.\_\_health) > 60:  
 self.\_\_health.popleft()  
 async with asyncio.Lock():  
 if self.\_\_with\_tunneling and self.\_\_tunnel.connected:  
 event = (TunnelEvent.WafHealthUpdate, list(self.\_\_health))  
 await self.\_\_tunnel.register\_event(pickle.dumps(event))  
  
 @register\_event(event=TunnelEvent.AccessLogUpdate)  
 async def \_\_handle\_authorized\_client(self, client: Connection, event: Event):  
 *"""  
 Handles an authorized client.  
 :param client:  
 :param event:  
 :return:  
 """* # Update the profile.  
 with self.\_\_profile\_manager as pm:  
 profile\_updates = {  
 "last\_used\_port": client.port,  
 "last\_connection\_time": event.log.sys\_epoch\_time,  
 "last\_event": event  
 }  
 pm.update\_profile(client.hash, profile\_updates)  
  
 # Connect to the web server.  
 stream = await AsyncStream.open\_stream(\*self.\_\_target)  
 web\_server = Connection(stream=stream, addr=HostAddress(\*self.\_\_target))  
  
 # Forward the clients request.  
 await forward\_data(web\_server, event.request.raw)  
 response = await recv\_from\_server(web\_server)  
 await forward\_data(client, response)  
 self.\_\_event\_manager.cache\_event(event)  
  
 @register\_event(TunnelEvent.SecurityLogUpdate)  
 async def \_\_handle\_unauthorized\_client(self, client: Connection, event: Event):  
 *"""  
 Handles an event made by an unauthorized user.  
 :param event:  
 :return:  
 """* # Update the profile.  
 with self.\_\_profile\_manager as pm:  
 profile: Profile = pm.get\_profile\_by\_hash(client.hash)  
 profile\_updates = {  
 "last\_used\_port": client.port,  
 "last\_connection\_time": get\_epoch\_time(),  
 "last\_event": event,  
 "attempted\_attacks": profile.attempted\_attacks + 1,  
 "last\_attempted\_attack": ",".join(event.log.classifiers)  
 }  
 pm.update\_profile(client.hash, profile\_updates)  
  
 # If this \_\_handle is called, then client is not banned.  
 # Ban the client and set correct duration.  
 profile = self.\_\_profile\_manager.get\_profile\_by\_hash(client.hash)  
  
 if profile.attempted\_attacks > self.\_\_config.banning["threshold"]: # exceeded the threshold.  
 banned\_at = get\_epoch\_time()  
 ban\_duration = profile.attempted\_attacks \* self.\_\_config.banning["factor"]  
 self.\_\_ban\_manager.insert\_mapping(client.hash, banned\_at, float(ban\_duration))  
  
 # Build a security page.  
 further\_information = ""  
 security\_page\_header = ""  
 match event.log.classifiers:  
 case [Classifier.SqlInjection | Classifier.XSS | Classifier.UnauthorizedAccess | Classifier.BannedAccess]:  
 security\_page\_header = self.\_\_config.securitypage["attack\_header"]  
 further\_information = self.\_\_config.securitypage["attack\_additional\_info"]  
 case [Classifier.Anonymity]:  
 security\_page\_header = self.\_\_config.securitypage["anonymity\_header"]  
 further\_information = self.\_\_config.securitypage["anonymity\_additional\_info"]  
 case [Classifier.BannedGeolocation]:  
 security\_page\_header = self.\_\_config.securitypage["geo\_header"]  
 further\_information = self.\_\_config.securitypage["geo\_additional\_info"]  
 case [Classifier.Anonymity, Classifier.BannedGeolocation]:  
 security\_page\_header = self.\_\_config.securitypage["dirty\_header"]  
 further\_information = self.\_\_config.securitypage["dirty\_additional\_info"]  
  
 self.\_\_event\_manager.cache\_event(event)  
 security\_page: bytes = create\_security\_page(info={  
 "header": security\_page\_header,  
 "further\_information": further\_information,  
 "token": event.id  
 })  
 await forward\_data(client, security\_page)  
  
 @register\_event(TunnelEvent.WafServicesUpdate)  
 async def \_\_handle\_connection(self, reader: asyncio.StreamReader, writer: asyncio.StreamWriter) -> None:  
 *"""  
 Handles each request by a single client.  
 :param reader:  
 :param writer:  
 :return: None  
 """* # Change the IP of the connection to the external ip of the Waf instance.  
 # This change happens only when the Waf instance is configured to be a local cluster.  
 ip, port = writer.get\_extra\_info(REMOTE\_ADDR)  
 if self.\_\_local:  
 ip = get\_external\_ip()  
  
 client = Connection(stream=AsyncStream(reader, writer), addr=HostAddress(ip, port))  
 self.\_\_set\_connection\_profile(client)  
  
 if self.\_\_ban\_manager.banned(client.hash):  
 security\_log = SecurityLog(  
 ip=client.ip,  
 port=client.port,  
 download=True,  
 sys\_epoch\_time=get\_epoch\_time(),  
 creation\_date=get\_unix\_time(self.\_\_config.timezone["time\_zone"]),  
 classifiers=[Classifier.BannedAccess],  
 geolocation=get\_geoip\_data(client.ip)  
 )  
 event = Event(  
 kind=UNAUTHORIZED\_REQUEST,  
 id=tokenize(),  
 log=security\_log,  
 request=None,  
 response=None,  
 )  
 await self.\_\_handle\_unauthorized\_client(client, event)  
 return  
  
 access\_list = self.\_\_acl  
 banned\_countries = self.\_\_config.geoip["banned\_countries"]  
  
 flags = validate\_dirty\_client(client.ip, access\_list, banned\_countries)  
  
 if flags != 0:  
 log\_classifiers = classify\_by\_flags(flags)  
 security\_log = SecurityLog(  
 ip=client.ip,  
 port=client.port,  
 download=True,  
 sys\_epoch\_time=get\_epoch\_time(),  
 creation\_date=get\_unix\_time(self.\_\_config.timezone["time\_zone"]),  
 classifiers=log\_classifiers,  
 geolocation=get\_geoip\_data(client.ip)  
 )  
 event = Event(  
 kind=UNAUTHORIZED\_REQUEST,  
 id=tokenize(),  
 log=security\_log,  
 request=None,  
 response=None,  
 )  
 await self.\_\_handle\_unauthorized\_client(client, event)  
 return  
  
 request = await recv\_from\_client(client, execption\_callback=client.close)  
 tx: Transaction = Transaction(  
 owner=client.addr,  
 real\_host\_address=None,  
 raw=request,  
 creation\_date=get\_unix\_time(self.\_\_config.timezone["time\_zone"])  
 )  
 tx.process()  
  
 access\_list = self.\_\_acl  
 banned\_countries = self.\_\_config.geoip["banned\_countries"]  
 check\_result = await check\_transaction(tx, access\_list, banned\_countries)  
  
 if check\_result.result:  
 security\_log = SecurityLog(  
 ip=client.ip,  
 port=client.port,  
 download=True,  
 sys\_epoch\_time=get\_epoch\_time(),  
 creation\_date=get\_unix\_time(self.\_\_config.timezone["time\_zone"]),  
 classifiers=check\_result.classifiers,  
 geolocation=get\_geoip\_data(client.ip)  
 )  
 event = Event(  
 kind=UNAUTHORIZED\_REQUEST,  
 id=tokenize(),  
 log=security\_log,  
 request=tx,  
 response=None,  
 )  
 await self.\_\_handle\_unauthorized\_client(client, event)  
 return  
  
 access\_log = AccessLog(  
 ip=client.ip,  
 port=client.port,  
 download=True,  
 geolocation=get\_geoip\_data(client.ip),  
 sys\_epoch\_time=get\_epoch\_time(),  
 creation\_date=get\_unix\_time(self.\_\_config.timezone["time\_zone"])  
 )  
 event = Event(  
 kind=AUTHORIZED\_REQUEST,  
 id=str(tx.hash),  
 log=access\_log,  
 request=tx,  
 response=None  
 )  
 await self.\_\_handle\_authorized\_client(client, event)  
  
 @classmethod  
 def get\_health(cls) -> float:  
 *"""Returns the current CPU usage."""* return psutil.cpu\_percent(1)  
  
 def get\_services\_report(self):  
 *"""Returns the report of the services."""* waf\_report = self.waf\_report  
 redis\_report = self.\_\_event\_manager.service\_report  
 sqlite\_report = self.\_\_profile\_manager.service\_report  
 return redis\_report | sqlite\_report | waf\_report  
  
 def get\_authorized\_events(self) -> list:  
 *"""Retrieves all the authorized events until this point in time."""* return self.\_\_event\_manager.get\_access\_events()  
  
 def get\_security\_events(self) -> list:  
 *"""Retrieves all the unauthorized events until this point in time."""* return self.\_\_event\_manager.get\_security\_events()  
  
 def get\_attack\_distribution(self):  
 *"""Returns a dictionary that maps a vulnerability with its distribution."""*  
 distributions = [event.log.classifiers[0] for event in self.get\_security\_events()]  
 return {classifier: distributions.count(classifier) for classifier in Classifier}  
  
 def start\_acl\_loop(self):  
 *"""Starts the Waf ACL loop."""* self.\_\_acl.activity\_loop()  
  
 def start\_cpu\_loop(self):  
 *"""Starts the health monitor for the frontend."""* if not self.\_\_with\_tunneling:  
 return  
 asyncio.run(self.\_\_cpu\_loop())  
  
 @register\_event(TunnelEvent.WafServicesUpdate)  
 async def deploy(self) -> None:  
 *"""  
 Creates an Asyncio.Server with corresponding configurations.  
 Note:  
 Waf.deploy() \*DOES NOT\* run the server.  
 :return: None  
 """* if self.\_\_state is \_WafState.DEPLOYED:  
 raise StateError("Instance already deployed")  
 if self.\_\_state is \_WafState.WORKING:  
 raise StateError("Instance is working, try Waf.close() then Waf.deploy()")  
 if self.\_\_state is \_WafState.CLOSED:  
 raise StateError("Instance is closed, try Waf.restart()")  
  
 # Deploy instance, state is \_Waf.CREATED  
 self.\_\_server: asyncio.Server = await asyncio.start\_server(  
 client\_connected\_cb=self.\_\_handle\_connection,  
 host=self.\_\_config.waf["ip"],  
 port=self.\_\_config.waf["port"],  
 start\_serving=False  
 )  
 self.\_\_state = \_WafState.DEPLOYED  
 print(f"INFO: Waf deployed AT: {self.\_\_address!s}, FOR: {self.\_\_target!s}")  
  
 @register\_event(TunnelEvent.WafServicesUpdate)  
 async def work(self):  
 *"""  
 Starts the main working task of the Waf - making it available to handle connections.  
 Note:  
 Waf serves forever and will stop when the coroutine is cancelled.  
 :return: None  
 """* if self.\_\_state is \_WafState.CREATED:  
 raise StateError("Instance is created, try Waf.deploy() before running")  
 if self.\_\_state is \_WafState.WORKING:  
 raise StateError("Instance is working")  
 if self.\_\_state is \_WafState.CLOSED:  
 raise StateError("Instance is closed, try Waf.restart()")  
  
 # Run instance, state is \_WafState.DEPLOYED.  
 async with self.\_\_server:  
 acl\_thread = create\_new\_thread(func=self.start\_acl\_loop, args=(), daemon=True)  
 cpu\_thread = create\_new\_thread(func=self.start\_cpu\_loop, args=(), daemon=True)  
  
 acl\_thread.start()  
 cpu\_thread.start()  
  
 work = [  
 create\_new\_task(task\_name="WAF\_WORKER", task=self.\_\_server.serve\_forever, args=()),  
 create\_new\_task(task\_name="WAF\_TUNNEL\_CONNECT", task=self.\_\_tunnel.connect, args=())  
 ]  
  
 # Start serving.  
 print(f"INFO: Waf listening AT {self.\_\_address}")  
 self.\_\_state = \_WafState.WORKING  
 self.\_\_deploy\_time = get\_unix\_time(self.\_\_config.timezone["time\_zone"])  
 await asyncio.gather(\*work if self.\_\_with\_tunneling else work[0])  
  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

class TunnelEvent(StrEnum):  
 TunnelConnection = 'tunnel\_connection'  
 TunnelDisconnection = 'tunnel\_disconnection'  
  
 AccessLogUpdate = 'access\_log\_update'  
 SecurityLogUpdate = 'security\_log\_update'  
  
 AttackDistributionUpdate = 'attack\_distribution\_update'  
 WafHealthUpdate = 'waf\_health\_update'  
 WafServicesUpdate = 'waf\_services\_update'  
  
  
class Tunnel:  
 *"""  
 A tunnel for communicating with the admin backend server.  
 This tunnel will notify the admin of every event of the Waf system.  
 """* def \_\_init\_\_(self, endpoint: str):  
 self.\_\_connected = False  
 self.\_\_websocket = None  
 self.\_\_event\_queue = asyncio.Queue()  
 self.\_\_tunnel\_endpoint = endpoint  
  
 @property  
 def endpoint(self) -> str:  
 return self.\_\_tunnel\_endpoint  
  
 @property  
 def connected(self) -> bool:  
 return self.\_\_connected  
  
 async def connect(self, timeout=10):  
 *"""  
 Connects to the endpoint.  
 Executes all events that were published in the connection process.  
 It does that by saving them in a queue and registers them one by one in order.  
 """* async def \_connect():  
 while True:  
 try:  
 return connect(self.\_\_tunnel\_endpoint)  
 except websockets.exceptions.InvalidHandshake:  
 break  
 except ConnectionRefusedError:  
 """Server is not up so the tunnel waits for him."""  
 await asyncio.sleep(0.5)  
 try:  
 self.\_\_websocket = await asyncio.wait\_for(asyncio.create\_task(\_connect()), timeout=timeout)  
 self.\_\_connected = True  
 await self.register\_event(pickle.dumps((TunnelEvent.TunnelConnection, '')))  
 while not self.\_\_event\_queue.empty():  
 await self.register\_event(await self.\_\_event\_queue.get())  
 print(f"Tunnel is connected to {self.\_\_tunnel\_endpoint}")  
 except asyncio.TimeoutError:  
 print("Timeout for tunnel connection has ended.")  
 return  
  
 async def reconnect(self):  
 *"""Reconnects to the endpoint."""* if self.\_\_websocket is not None or not self.\_\_connected:  
 raise StateError("Tunnel is open.")  
 await self.connect()  
  
 async def register\_event(self, event: bytes):  
 *"""Sends a tunnel event to the endpoint."""* try:  
 if not self.connected:  
 await self.\_\_event\_queue.put(event)  
 return  
 self.\_\_websocket.send(event)  
 except websockets.exceptions.ConnectionClosedError:  
 """Tunnel is closed."""  
 return  
  
 def recv\_result(self) -> EventResult:  
 *"""Returns the data sent from the websocket server."""* return self.\_\_websocket.recv()  
  
 def register\_recv(self, event) -> EventResult:  
 *"""Performs a send/recv operation."""* self.register\_event(event)  
 return self.recv\_result()

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
@dataclass(slots=True)  
class AccessList:  
 *"""A class representing a list of ip addresses of untrusted proxies."""* main\_list: list  
 api: str  
 interval: int  
 backup: str  
  
 def fetch\_anonymous\_proxies(self):  
 try:  
 response = requests.get(self.api)  
 if response.status\_code != 200:  
 raise AclFetchError(f"ACL.{AccessList.fetch\_anonymous\_proxies.\_\_name\_\_} failed")  
 self.main\_list = response.text.split("\n")  
  
 except (AclFetchError, requests.exceptions.SSLError):  
 try:  
 with open(self.backup, "r") as exit\_nodes:  
 self.main\_list = exit\_nodes.read().split("\n")  
 except FileNotFoundError:  
 raise AclBackUpError("Backup is not available. Please check config.toml for ACL.backup")  
  
 def activity\_loop(self, max\_retries=10):  
 *"""Activity loop of refetching the Tor exit nodes."""* tries = 0  
 while True:  
 try:  
 self.fetch\_anonymous\_proxies()  
 time.sleep(self.interval)  
 except (AclFetchError, AclBackUpError) as e:  
 if tries <= max\_retries:  
 tries += 1  
 print(f"{e}, retrying")  
 continue  
 break  
 except asyncio.exceptions.CancelledError:  
 """Coroutine was cancelled."""  
 return  
 raise Exception("Reached loop limit, check for API connection")  
  
 def \_\_contains\_\_(self, ip):  
 return ip in self.main\_list

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

class BanManager(metaclass=Singleton):  
 *"""  
 A wrapper class to redis.Redis representing a ban manager.  
  
 Notes  
 -----  
 1. BanManager will access the redis db in a separate task evaluating all the bans.  
 2. A client is freed when a ban is over or when the admin removes the ban.  
 3. A ban is extended when a client exceeds the attack threshold.  
 """* def \_\_init\_\_(self, host="localhost", ban\_namespace="bans", port=6379, logs=True):  
 try:  
 self.\_\_logs = logs  
 self.\_\_ban\_namespace = ban\_namespace  
 self.r = redis.Redis(host=host, port=port, decode\_responses=True)  
 except redis.exceptions.ConnectionError:  
 print("Redis server is not running. Please run scripts/run\_redis before deploying.")  
  
 def log(self, msg: str):  
 *"""  
 Log `msg` if \_\_logs is True.  
 :param msg:  
 :return:  
 """* if self.\_\_logs:  
 print(msg)  
  
 def redis\_alive(self) -> bool:  
 *"""  
 Checks the connection to the redis server.  
 :return:  
 """* return self.r.ping()  
  
 def insert\_mapping(self, client\_hash: str, banned\_at: float, ban\_duration: float):  
 *"""  
 Sets a new entry inside the cache of client\_hash and its ban state.  
 Will expire the cache entry automatically if not configured otherwise.  
 :param client\_hash:  
 :param ban\_duration:  
 :param banned\_at:  
 :return:  
 """* self.r.hset(f'{self.\_\_ban\_namespace}:{client\_hash}', key="banned\_at", value=str(banned\_at))  
 self.r.hset(f'{self.\_\_ban\_namespace}:{client\_hash}', key="duration\_remaining", value=str(ban\_duration))  
 self.r.expire(f'{self.\_\_ban\_namespace}:{client\_hash}', time=int(ban\_duration))  
  
 def get\_mapping(self, client\_hash: str) -> dict:  
 *"""  
 Retrieves an entry from the cache.  
 :param client\_hash:  
 :return:  
 """* if not self.r.exists(client\_hash):  
 self.log(f"Given argument {client\_hash} does not exist in the cache")  
 return {}  
 return self.r.hgetall(client\_hash)  
  
 def remove\_client(self, client\_hash: str):  
 *"""  
 Removes a client\_hash and its mapping from the cache.  
 :param client\_hash:  
 :return:  
 """* if not self.r.exists(client\_hash):  
 self.log(f"Given argument {client\_hash} does not exist in the cache")  
 self.r.hdel(client\_hash)

def banned(self, client\_hash: str) -> bool:  
 *"""  
 Returns if the client\_hash represents an active ban.  
 :param client\_hash:  
 :return:  
 """* return self.r.exists(f'{self.\_\_ban\_namespace}:{client\_hash}')  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

class EventManager(metaclass=Singleton):  
 def \_\_init\_\_(  
 self,  
 host="localhost",  
 port=6379,  
 access\_events\_namespace="access\_events",  
 security\_events\_namespace="security\_events",  
 logs=True  
 ):  
 try:  
 self.\_\_logs = logs  
 self.\_\_host, self.\_\_port = host, port  
 self.\_\_last\_security\_update, self.\_\_last\_access\_update = 'Nan', 'Nan'  
 self.\_\_access\_events\_namespace = access\_events\_namespace  
 self.\_\_security\_events\_namespace = security\_events\_namespace  
 self.access\_events\_redis = redis.Redis(host=host, port=port)  
 self.security\_events\_redis = redis.Redis(host=host, port=port)  
 except redis.exceptions.ConnectionError:  
 print("Redis server is not running. Please run scripts/run\_redis before deploying.")  
  
 @property  
 def service\_report(self):  
 *"""Builds a report of the EventManager."""* access\_cache\_size, security\_cache\_size = len(self.get\_access\_events()), len(self.get\_security\_events())  
 return {  
 "redis\_host": self.\_\_host,  
 "redis\_port": self.\_\_port,  
 "cache\_size": access\_cache\_size + security\_cache\_size,  
 "access\_events\_size": access\_cache\_size,  
 "security\_events\_size": security\_cache\_size,  
 "last\_security\_update": self.\_\_last\_security\_update,  
 "last\_access\_update": self.\_\_last\_access\_update  
 }  
  
 def log(self, msg: str):  
 *"""  
 Log `msg` if \_\_logs is True.  
 :param msg:  
 :return:  
 """* if self.\_\_logs:  
 print(msg)

def redis\_alive(self) -> tuple:  
 *"""  
 Checks the connections to the redis server.  
 :return:  
 """* return self.access\_events\_redis.ping(), self.security\_events\_redis.ping()  
  
 def cache\_event(self, event: Event):  
 *"""  
 Sets a new entry inside the cache of client\_hash and its access log.  
 Will expire the cache entry automatically if not configured otherwise.  
 :return:  
 """* redis\_client, namespace = None, ''  
   
 if event.kind == AUTHORIZED\_REQUEST:  
 redis\_client = self.access\_events\_redis  
 namespace = self.\_\_access\_events\_namespace  
 self.\_\_last\_access\_update = get\_unix\_time("Asia/Jerusalem")  
  
 if event.kind == UNAUTHORIZED\_REQUEST:  
 redis\_client = self.security\_events\_redis  
 namespace = self.\_\_security\_events\_namespace  
 self.\_\_last\_security\_update = get\_unix\_time("Asia/Jerusalem")  
  
 redis\_client.zadd(namespace, mapping={Event.serialize(event): event.log.sys\_epoch\_time})  
  
 def get\_access\_events(self):  
 *"""Retrieves the access, events ordered set from the cache."""* return [Event.deserialize(event) for event in  
 self.access\_events\_redis.zrange(self.\_\_access\_events\_namespace, start=0, end=-1)]  
  
 def get\_security\_events(self):  
 *"""Retrieves the security events, ordered set from the cache."""* return [Event.deserialize(event) for event in  
 self.security\_events\_redis.zrange(self.\_\_security\_events\_namespace, start=0, end=-1)]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

class ProfileManager(metaclass=Singleton):  
 *"""  
 A class for handling profiles and client activities.  
 """* \_\_db\_path = ROOT\_PATH / "profiles\_db" / "database.db"  
  
 def \_\_init\_\_(self):  
 conn = sqlite3.connect(ProfileManager.\_\_db\_path)  
 conn.cursor().execute("""  
 CREATE TABLE IF NOT EXISTS profiles (ip\_hash TEXT PRIMARY KEY, profile\_object BLOB)  
 """)  
 self.conn = conn  
 self.cursor = self.conn.cursor()  
  
 @property  
 def service\_report(self):  
 *"""Returns a report of the manager."""* return {}  
  
 def insert\_profile(self, client\_hash: str, profile: Profile) -> None:  
 *"""  
 Inserts a profile mapped by the hash into the db.  
 :param client\_hash: str  
 :param profile:  
 :return:  
 """* params = (client\_hash, profile.serialize())  
 self.cursor.execute("INSERT OR IGNORE INTO profiles VALUES(?, ?)", params)  
  
 def get\_profile\_by\_hash(self, client\_hash: str) -> Union[Profile, None]:  
 *"""  
 Fetches the profile attached to the ip hash.  
 :param client\_hash:  
 :return:  
 """* self.cursor.execute("SELECT profile\_object FROM profiles WHERE ip\_hash=?", (client\_hash,))  
 profile = self.cursor.fetchone()  
 if profile is not None:  
 return Profile.deserialize(profile[0])  
 return None  
  
 def delete\_profile(self, client\_hash: str):  
 *"""  
 Deletes a specific profile from the db.  
 :param client\_hash:  
 :return:  
 """* self.cursor.execute("DELETE FROM profiles WHERE ip\_hash=?", (client\_hash,))  
  
 def update\_profile(self, client\_hash: str, kwargs: dict):  
 *"""  
 Updates a profile with kwargs, where kwargs holds fields and values.  
 :param client\_hash:  
 :param kwargs:  
 :return:  
 """* profile: Profile = self.get\_profile\_by\_hash(client\_hash)  
 profile.update(kwargs)  
 updated\_row = (profile.serialize(), client\_hash)  
 self.cursor.execute("UPDATE profiles SET profile\_object=? WHERE ip\_hash=?", updated\_row)

def get\_all\_profiles(self) -> dict[bytes, Profile]:  
 *"""  
 Creates a dictionary of all profiles in the database.  
 :return: dict of profiles.  
 """* self.cursor.execute("SELECT \* FROM profiles")  
 return {ip\_hash: Profile.deserialize(profile) for ip\_hash, profile in self.cursor.fetchall()}  
  
 def \_\_enter\_\_(self):  
 return self  
  
 def \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):  
 self.conn.commit()

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ROOT\_DIR = Path(\_\_file\_\_).parent  
TEMPLATES\_PATH = ROOT\_DIR / "templates"  
BLOCK\_REGEX: re.Pattern[str] = re.compile(r"/block[?]token=([a-z0-9]{8})")  
  
  
def create\_redirection(location: bytes) -> bytes:  
 redirect = b"HTTP/1.1 302 Found\r\n"  
 redirect += b"Location: " + location + b"\r\n\r\n"  
 return redirect  
  
  
def contains\_block(tx: Transaction) -> str | None:  
 *"""  
 Uses regex to get the token of a transaction seeking its security page.  
 :param tx:  
 :return:  
 """* resource: bytes = urlunparse(tx.url)  
 m: re.Match[str] = re.match(BLOCK\_REGEX, resource)  
 if not m:  
 return None  
 return m.group(1)

class TemplateEnv(Environment, metaclass=Singleton):  
 *"""  
 A wrapper class to jinja2.Environment.  
 """* def \_\_init\_\_(self):  
 super().\_\_init\_\_(loader=FileSystemLoader(TEMPLATES\_PATH))  
  
 def render\_kwargs(self, \*\*kwargs) -> bytes:  
 *"""  
 Creates a complete template with the given kwargs.  
 :param kwargs:  
 :return:  
 """* general\_template = self.get\_template("general\_template.html")  
 return general\_template.render(\*\*kwargs).encode("utf-8")

def create\_security\_page(info):  
 *"""  
 Builds a security page based on parameters inside info.  
 :param info:  
 :return:  
 """* html: bytes = TemplateEnv().render\_kwargs(\*\*info)  
 content\_length = str(len(html)).encode("utf-8")  
 response = b"HTTP/1.1 200 OK\r\n"  
 response += b"Content-Type: text/html; charset=utf-8\r\n"  
 response += b"Content-Length: " + content\_length + b"\r\n"  
 response += b"Connection: close\r\n\r\n"  
 response += html  
 return response  
  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ROOT\_FOLDER = "signatures"  
  
  
def abs\_node\_path(node: str, \*other\_nodes) -> Path:  
 *"""  
 Computes the absolute path of node (based on this root dir) joining each given node.  
 :returns: absolute path  
 """* this\_root = Path(\_\_file\_\_).parent  
 for n in other\_nodes:   
 this\_root = this\_root.joinpath(n)  
 absolute\_path = this\_root.joinpath(node)  
 return absolute\_path  
  
  
def \_load\_unauthorized\_data() -> list:  
 def \_filter\_data(\_data: str) -> bool:  
 return not \_data.startswith("#") and \_data  
  
 path = abs\_node\_path(ROOT\_FOLDER).joinpath("unauthorized\_access.txt")  
 with open(path, "r") as unauthorized\_locations:  
 return list(filter(\_filter\_data, unauthorized\_locations.read().splitlines()))  
  
  
def \_load\_json\_file(json\_file: str) -> dict:  
 path: Path = Path()  
 try:  
 path = abs\_node\_path(ROOT\_FOLDER).joinpath(json\_file)  
 with open(path, "r") as json\_data:  
 return json.load(json\_data)  
 except FileNotFoundError:  
 raise FileNotFoundError(  
 f"{json\_file} not found at {path}. Double check the given file."  
 )  
  
  
class DbState(Enum):  
 EMPTY = "Empty"  
 LOADED = "Loaded"  
  
  
class \_BaseDb:  
 \_state: DbState = DbState.EMPTY  
  
 def \_\_init\_\_(self) -> None:  
 if \_BaseDb.\_state is DbState.LOADED:  
 return  
 self.sql\_data\_set: dict = \_load\_json\_file("sql\_data.json")  
 self.xss\_data\_set: dict = \_load\_json\_file("xss\_data.json")  
 self.unauthorized\_access\_data\_set: list = \_load\_unauthorized\_data()  
 \_BaseDb.\_state = DbState.LOADED  
  
 @classmethod  
 def get\_state(cls):  
 return cls.\_state  
  
 @classmethod  
 def set\_state(cls, state: DbState):  
 if not isinstance(state, DbState):  
 raise ValueError(  
 f"Invalid argument type for {cls.set\_state.\_\_name\_\_}. Expected DbState, got {type(state).\_\_name\_\_}."  
 )  
 cls.\_state = state  
  
---------------------------------------------------------------------------

SQL\_INJECTION = 0x00000001 # 1  
XSS = 0x00000010 # 2  
LFI = 0x00000100 # 4  
RFI = 0x00001000 # 8  
BRUTEFORCE = 0x00010000 # 16  
UNAUTHORIZED\_ACCESS = 0x00100000 # 32  
ANONYMOUS = 0x01000000 # 64  
GEOLOCATION = 0x10000000 # 128  
  
  
@dataclass(slots=True)  
class Check:  
 *"""  
 A class representing a check to be done.  
 """* fn: Callable  
 args: tuple  
  
  
@dataclass(slots=True)  
class CheckResult:  
 *"""  
 A class representing a result of a check.  
  
 Attributes  
 ----------  
 result - boolean indicating if the check passed or not.  
 classifiers - If an attack is detected, the check function will return the classifier of the attack.  
 """* result: bool  
 classifiers: list[Classifier]

@dataclass(slots=True)  
class Profile:  
 *"""  
 A class for representing the active profile of a client.  
 """* host: str  
 connection\_date: str  
 last\_used\_port: int  
 last\_connection\_time: float  
 last\_event: Event  
 attempted\_attacks: int  
 last\_attempted\_attack: str  
  
 def update(self, data):  
 for setting, value in data.items():  
 setattr(self, setting, value)  
  
 def serialize(self) -> bytes:  
 return pickle.dumps(self)  
  
 @classmethod  
 def deserialize(cls, data: bytes):  
 return pickle.loads(data)  
  
---------------------------------------------------------------------------

@dataclass(slots=True)  
class Log:  
 *"""A basic class holding the basic fields that either a Security Log and an Access Log hold."""* ip: str  
 port: int  
 download: bool  
 sys\_epoch\_time: float  
 creation\_date: str  
 geolocation: GeoData | None  
  
 def serialize(self) -> bytes:  
 return pickle.dumps(self)  
  
  
@dataclass(slots=True)  
class SecurityLog(Log):  
 *"""A log object that is created when a Waf instance detected a potential attack or threat."""* classifiers: list[Classifier]  
 malicious\_data: Optional[bytes] = None  
 metadata: Optional[dict] = None  
  
  
@dataclass(slots=True)  
class AccessLog(Log):  
 *"""A log object representing valid client transactions."""* pass  
  
  
type LogObject = Union[SecurityLog, AccessLog]

---------------------------------------------------------------------------

class ToJson:  
 *"""A class that turns objects that are not serializable, serializable"""* @classmethod  
 def transaction(cls, tx: Transaction) -> Union[Transaction, None]:  
 if tx is None:  
 return None  
 return Transaction(  
 owner=tx.owner,  
 real\_host\_address=tx.real\_host\_address,  
 raw=tx.raw.decode("utf-8"),  
 creation\_date=tx.creation\_date,  
 method=tx.method.decode("utf-8"),  
 url=tx.url,  
 version=tx.version.decode("utf-8"),  
 query\_params=tx.query\_params,  
 headers=decode\_headers(tx.headers),  
 body=tx.body,  
 size=tx.size  
 )  
  
 @classmethod  
 def event(cls, event: Event):  
 temp = Event(kind=event.kind, id=event.id, log=event.log,  
 request=cls.transaction(event.request),  
 response=cls.transaction(event.response))  
 return json.dumps(asdict(temp))  
  
 def \_\_call\_\_(self, jsonable: object):  
 if isinstance(jsonable, Transaction):  
 return self.transaction(jsonable)  
 elif isinstance(jsonable, Event):  
 return self.event(jsonable)  
 else:  
 raise Exception(f"{jsonable} of type {type(jsonable)} is not Jsonable for the Jsonizer.")

---------------------------------------------------------------------------  
  
ROOT\_DIR = Path(\_\_file\_\_).parent  
\_\_mmdb\_path = ROOT\_DIR / "geoip2\_db" / "GeoLite2-City.mmdb"  
  
  
@dataclass(slots=True)  
class GeoData:  
 continent: str  
 country: str  
 city: str  
  
 def \_\_repr\_\_(self) -> str:  
 return f"{self.continent}, {self.country} - {self.city}"  
  
  
def get\_external\_ip() -> str:  
 *"""Uses the API of ipify.org to fetch the external IP of the cluster."""* response = requests.get("https://api.ipify.org?format=json").text  
 return json.loads(response)["ip"]  
  
  
def get\_geoip\_data(ip: str) -> GeoData | None:  
 *"""Gets the geoip data regarding an ip address."""* try:  
 with geoip2.database.Reader(\_\_mmdb\_path) as reader:  
 response = reader.city(ip)  
 return GeoData(  
 continent=response.continent.name,  
 country=response.country.name,  
 city=response.city.name,  
 )  
 except geoip2.errors.AddressNotFoundError:  
 return None  
  
  
def validate\_geoip\_data(ip: str, banned\_countries: list) -> bool:  
 geodata = get\_geoip\_data(ip)  
 if geodata is None:  
 # client is not found on the db, assuming he is not valid.  
 return False  
 return geodata.country in banned\_countries

---------------------------------------------------------------------------

class Classifier(StrEnum):  
 *"""  
 Classifier is a part of the classification process of each Security Log. It tells the Waf why the event was triggered.  
 Additionally, a classifier is the vulnerability identifier and the rationale behind the BLOCK action.  
  
 Variants  
 -------------  
 Sql Injection - The transaction showed signs of SQL signatures.  
 XSS - The transaction showed signs of XSS signatures.  
 Unauthorized Access - The transactions `request-URI` or namely tx.url aims to access a forbidden location.  
 Banned Access - The transaction was made by an already banned client.  
 Banned Geolocation - The transaction was made by a client living in a banned geolocation.  
 Anonymity - The transactions owner identifies with a banned software such as TOR.  
 """* SqlInjection = "Sql Injection"  
 XSS = "Cross Site Scripting"  
 RFI = "Remote File Inclusion"  
 LFI = "Local File Inclusion"  
 UnauthorizedAccess = "Unauthorized Access"  
 BannedAccess = "Banned Access"  
 BannedGeolocation = "Banned Geolocation"  
 Anonymity = "Anonymization"

---------------------------------------------------------------------------

REMOTE\_ADDR = "peername"  
SOCKET = "socket"  
SOCKNAME = "sockname"  
  
  
@dataclass(slots=True)  
class HostAddress:  
 *"""  
 A class representing hosts on the network.  
 """* ip: str  
 port: int  
  
  
class AsyncStream:  
 *"""  
 A class representing an asynchronous stream (part of Connection class).  
 """* \_BUFFER\_SIZE = 8192  
  
 def \_\_init\_\_(self, reader: asyncio.StreamReader=None, writer: asyncio.StreamWriter=None):  
 self.\_\_reader = reader  
 self.\_\_writer = writer  
  
 async def write(self, data: bytes):  
 self.\_\_writer.write(data)  
 await self.\_\_writer.drain()  
  
 async def close(self):  
 self.\_\_writer.close()  
 await self.\_\_writer.wait\_closed()  
  
 @classmethod  
 async def open\_stream(cls, host: str, port: int) -> Self:  
 try:  
 reader, writer = await asyncio.open\_connection(host=host, port=port)  
 return cls(reader, writer)  
 except OSError:  
 print("ERROR: could not connect to the given ip and port.")  
 return cls(None, None)  
  
 def \_\_bool\_\_(self):  
 return self.\_\_reader is not None and self.\_\_writer is not None  
  
 def \_\_aiter\_\_(self):  
 return self  
  
 async def \_\_anext\_\_(self) -> bytes:  
 data = await self.\_\_reader.read(n=AsyncStream.\_BUFFER\_SIZE)  
 if not data:  
 raise StopAsyncIteration  
 return data  
  
  
def looper(func: Callable):  
 *"""Decorator for running important functions in their own event loop."""* @wraps(func)  
 def wrapper(\*args, \*\*kwargs) -> None:  
 loop = asyncio.new\_event\_loop()  
 asyncio.set\_event\_loop(loop)  
 loop.run\_until\_complete(func(\*args, \*\*kwargs))  
 loop.close()  
 return wrapper  
  
  
def create\_new\_thread(func: Callable, args: tuple, daemon: bool=False) -> threading.Thread:  
 *"""  
 Creates a local thread.  
 :returns: Thread.  
 """* return threading.Thread(target=func, args=args, daemon=daemon)  
  
  
def create\_new\_task(task\_name: str=None, task: Callable=None, args: tuple=()) -> asyncio.Task:  
 *"""  
 Creates a new task.  
 :returns: Task.  
 """* if not task\_name:  
 task\_name = f"TASK\_NAME\_{task.\_\_name\_\_}"  
 return asyncio.create\_task(name=task\_name, coro=task(\*args))  
  
  
def convert\_netloc(netloc: str) -> Union[Network\_Address, None]:  
 *"""  
 Converts a netloc to a \*real\* netloc from ipaddress and checks if it's valid.  
 :param netloc: either an ip\_address of ip\_address:port  
 :return:  
 """* try:  
 ip\_frag, sep, port = netloc.rpartition(":")  
 assert sep, AssertionError  
 return ipaddress.ip\_address(ip\_frag), int(port)  
  
 except AssertionError:  
 # No port was specified.  
 try:  
 return ipaddress.ip\_address(netloc), -1  
 except ValueError:  
 return None  
  
 except ValueError:  
 return None  
  
  
def hash\_by\_host(host: str) -> str:  
 *"""  
 Creates a hash from an ip address.  
 :param host:  
 :return:  
 """* if not convert\_netloc(host):  
 print(f"Hash\_by\_ip error. Expected a valid IP address, got {host}")  
 return hashlib.sha1(host.encode("utf-8")).hexdigest()

---------------------------------------------------------------------------

@dataclass(slots=True)  
class Connection:  
 *"""  
 A class representing a connection over the network.  
 """* stream: AsyncStream  
 addr: HostAddress  
  
 @property  
 def ip(self) -> str:  
 return self.addr.ip  
  
 @property  
 def port(self) -> int:  
 return self.addr.port  
  
 @property  
 def hash(self) -> str:  
 return hash\_by\_host(self.ip)  
  
 async def recv\_until(self, condition: Callable, args: tuple) -> bytes:  
 *"""  
 Recv data until a condition is met.  
 The first argument of `condition` is ALWAYS data.  
  
 This function is not a replacement of:  
  
 async for chunk in connection.stream:  
 data += chunk  
  
 Instead, Connection.recv\_until serves as a conditional recv behavior.  
 :return: bytes  
 """* data = b""  
 async for chunk in self.stream:  
 data += chunk  
 if condition(data, \*args):  
 break  
 return data  
  
 async def write(self, data: bytes):  
 *"""  
 Writes data to the stream.  
 :param data:  
 :return:  
 """* await self.stream.write(data)  
  
 def close(self):  
 *"""  
 Closes the stream.  
 :return:  
 """* self.stream.close()  
  
 def \_\_hash\_\_(self) -> int:  
 return hash(repr(self))

---------------------------------------------------------------------------

JSONIZER = ToJson()  
  
  
class FinalWallApi(Flask):  
 def \_\_init\_\_(self, \*args, \*\*kwargs) -> None:  
 super().\_\_init\_\_(\*args, \*\*kwargs)  
  
 self.\_\_logger = create\_logger(name="API")  
 self.\_\_socketIO = SocketIO(self, cors\_allowed\_origins="\*", logger=False, engineio\_logger=False)  
  
 self.\_\_logger.info("FinalWall API started.")  
  
 def start\_tunnel\_server(self):  
 *"""Starts tunnel listener."""* asyncio.run(self.tunnel\_thread())  
  
 def start\_tunnel\_thread(self):  
 *"""Starts the tunnel receiving thread."""* create\_new\_thread(func=self.start\_tunnel\_server, args=(), daemon=True).start()  
  
 @classmethod  
 def process\_event\_data(cls, event\_type: TunnelEvent, data):  
 *"""Processes the data of the event in order to forward it to the Frontend."""* if event\_type == TunnelEvent.TunnelConnection:  
 return get\_unix\_time("Asia/Jerusalem")  
 elif event\_type == TunnelEvent.AccessLogUpdate:  
 return [JSONIZER(\_event) for \_event in data]  
 elif event\_type == TunnelEvent.SecurityLogUpdate:  
 return {"events": [JSONIZER(\_event) for \_event in data["events"]], "distribution": data["distribution"]}  
 return data  
  
 async def tunnel\_thread(self):  
 *"""Starts the websocket tunnel server."""* async def \_\_tunnel\_event\_handler(websocket: WebSocketServerProtocol):  
 *"""Handler for the catching events from the main cluster."""* try:  
 async for event in websocket:  
 async with asyncio.Lock():  
 event\_type, data = pickle.loads(event)  
 processed\_data = FinalWallApi.process\_event\_data(event\_type, data)  
 self.\_\_socketIO.emit(event\_type, processed\_data)  
  
 except websockets.exceptions.ConnectionClosed:  
 """Can either be raised from poor connectivity or large amounts of traffic."""  
 return  
  
 async with serve(\_\_tunnel\_event\_handler, "localhost", 8765):  
 await asyncio.get\_event\_loop().create\_future()  
  
  
async def main() -> None:  
 *"""Main entry for the API."""* app = FinalWallApi(import\_name=\_\_name\_\_)  
 CORS(app, resources={r"/\*": {"origins": "\*"}})  
 app.start\_tunnel\_thread()  
  
 app.run(host="localhost", port=5001)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 import tracemalloc  
 tracemalloc.start()  
  
 import sys  
  
 if sys.version\_info[0:2] != (3, 12):  
 raise VersionError("Wrong python version. Please use +=3.12 only")  
  
 asyncio.run(main())

---------------------------------------------------------------------------  
  
# Author: Daniel Sapojnikov 2023.  
# config.toml provides configurations for WAF deployment and capabilities.  
  
### Network Configurations  
  
[WebServer]  
ip = "127.0.0.1"  
port = 50000  
  
[Waf]  
ip = "0.0.0.0"  
port = 8080  
  
  
[Admin]  
backend\_api = "ws://localhost:8765"  
  
### Further sections are related to WAF features.  
  
[ACL]  
api = "https://check.torproject.org/exit-addresses"  
backup = "C:\\coding\\FinalWall\\engine\\proxy\_network\\anonymity\\TorExitNodes.txt"  
interval = 1800 # 30 minutes  
  
[Banning]  
threshold = 3  
factor = 3600 # 1 hour.  
  
[GeoIP]  
banned\_countries = ["IR", "NK", "PS", "SY", "LB"]  
  
[Timezone]  
time\_zone = "Asia/Jerusalem"

### Further sections are related to security page information.  
  
[SecurityPage]  
  
# General attack info.  
attack\_header = "Wrongful behavior detected."  
attack\_additional\_info = ["Your activity log was sent to our administration team and is being investigated."]  
  
# Info regarding banned software.  
anonymity\_header = "We have detected usage of anonymization software."  
anonymity\_additional\_info = ["We suggest you try to connect to our services without identifying with anonymization products."]  
  
# Info regarding banned geopraphic locations.  
geo\_header = "We have detected illegal activity based on your geographic location."  
geo\_additional\_info = ["We suggest you try to connect to our services without identifying with anonymization products."]  
  
# For both geolocation and banned software.  
dirty\_header = "Anonymization software and a banned geolocation have been detected."  
dirty\_additional\_info = ["You are seeing this page because we have found your location to be untrusted",  
 "additionaly, we found anonymization attempt via anonymization software."]

----------------------------------------------------------------------------------

Frontend UI Store.js code:  
  
  
import Vuex from 'vuex'  
import axios from 'axios';  
import { Operation } from './ops';  
import Help from '@/views/Help.vue';  
  
const currentTimeFormatted = () => {  
 let now = new Date();  
 let options = {   
 day: '2-digit',   
 month: 'short',   
 year: 'numeric',   
 hour: '2-digit',   
 minute: '2-digit',   
 hour12: false  
 };  
 let dateString = now.toLocaleString('en-GB', options);  
 return dateString;  
};  
  
// Load the state from localStorage  
const loadStateFromLocalStorage = () => {  
 localStorage.removeItem('vuexState');  
 const state = localStorage.getItem('vuexState');  
 return state ? JSON.parse(state) : {  
 lastUpdate: '',  
 totalTransactions: 0,  
 allowedTransactions: 0,  
 blockedTransactions: 0,  
 accessEvents: [],  
 securityEvents: [],  
 health: [],  
 attackDistribution: {},  
 attackSources: {  
 "sources": [],  
 "numbers": []  
 },  
 services: {}  
 };  
};  
  
const store = new Vuex.Store({  
 state: loadStateFromLocalStorage(),  
 mutations: {  
 updateState(state) {  
 localStorage.setItem('vuexState', JSON.stringify(state));  
 },  
 updateTotalTransactions(state) {  
 state.totalTransactions = state.allowedTransactions + state.blockedTransactions  
 },  
 updateLatestUpdate(state) {  
 state.lastUpdate = currentTimeFormatted();  
 },  
 updateAccessEvents(state, events) {  
 state.accessEvents = events.map(event => { return JSON.parse(event) });  
 state.allowedTransactions = events.length  
 },  
 updateSecurityEvents(state, events) {  
 state.securityEvents = events.map(event => { return JSON.parse(event) });  
 state.blockedTransactions = events.length  
 },  
 updateAttackDistribution(state, scores) {  
 state.attackDistribution = scores  
 },  
 updateHealth(state, health) {  
 state.health = health  
 },  
 updateServices(state, services) {  
 state.services = services  
 },  
 updateDeployTime(state, time) {  
 state.deployedAt = time  
 },  
 updateTopAttackSources(state) {  
 state.attackSources = {}  
 state.securityEvents.forEach(event => {  
 if (!state.attackSources[event.log.ip]) {  
 state.attackSources[event.log.ip] = 0;  
 }  
 state.attackSources[event.log.ip]++;  
 });  
 const sortedDistributions = Object.keys(state.attackSources)  
 .map(ip => ({ ip: ip, times: state.attackSources[ip] }))  
 .sort((a, b) => b.times - a.times);  
 state.attackSources["sources"] = sortedDistributions.slice(0, 5).map((source) => source.ip);  
 state.attackSources["numbers"] = sortedDistributions.slice(0, 5).map((source) => source.times);  
 }  
 },  
 actions: {  
 updateAccessEvents({ commit }, { events }) {  
 commit('updateAccessEvents', events);  
 commit('updateTotalTransactions');  
 commit('updateLatestUpdate');  
 commit('updateState');  
 },  
 updateSecurityEvents({ commit }, { events, distribution }) {  
 commit('updateSecurityEvents', events);  
 commit('updateTotalTransactions');  
 commit('updateLatestUpdate');  
 commit('updateTopAttackSources')  
 commit('updateState');  
 this.dispatch('updateAttackDistribution', { distribution });  
 },  
 updateAttackDistribution({ commit }, { distribution }) {  
 commit('updateAttackDistribution', distribution);  
 commit('updateLatestUpdate');  
 commit('updateState');  
 },  
 updateHealth({ commit }, { health }) {  
 commit('updateHealth', health);  
 commit('updateState');  
 },  
 updateServices({ commit }, { services }) {  
 commit('updateServices', services);  
 commit('updateState');  
 }  
 }  
});  
  
export default store